



TECNOLÓGICO
NACIONAL DE MÉXICO



INSTITUTO TECNOLÓGICO DE CHIHUAHUA

MANUAL DEL ROBOT SERVICIO SOCIAL

Robot Hiwonder ArmPi Pro basado en Raspberry Pi 4B

controlado de forma remota.

Docente/Supervisor: Ing. Alfredo Chacón Almada

Participantes:

Marco Antonio Calderón Macías

20060915

Alejandro Morales Holguín

20060938

Contenido

1.	Robot ArmPi Pro de Hiwonder.....	4
2.	Sistema operativo Linux – Ubuntu 18.04.....	5
2.1.	Conexión vía SSH con el sistema.....	5
2.2.	Softwares de control desarrollados por Hiwonder.....	7
2.2.1.	ArmPi Pro PC.....	7
2.2.2.	LAB_Tool	11
2.3.	Actualización e instalación de paquetes del sistema operativo.	12
2.3.1.	Instalación de nuevos paquetes.	12
2.4.	Inicialización de servicios desde el boot de Linux.	13
3.	Control del ArmPi Pro por medio de Python.	18
3.1.	Control del chasis de ruedas omnidireccionales.....	18
3.2.	Control de los servomotores del brazo robótico.....	19
3.3.	Control de la cámara USB.....	19
3.3.1.	Módulo de cámara de ROS.....	19
3.3.2.	Transmisión y visión artificial con Flask y OpenCV.....	21
4.	Aplicación de control en Visual Studio 2019.....	24
4.1.	Interfaz gráfica y elementos básicos.....	24
4.2.	Conexión del control remoto e interpretación de los botones.	25
4.2.1.	Instalación de DS4 Windows.	25
4.2.2.	Instalación de la librería XInput.....	26

4.2.3.	Uso de la librería XInput.....	29
4.3.	Comunicación entre aplicación y los servidores.	30
4.3.1.	Python (Flask).	30
4.3.2.	Visual Studio	31
4.4.	Conexión con el servidor de la cámara.....	34
4.4.1.	Configuración de la cámara usando qv4l2.	34
4.5.	Obtención de los valores de batería (JSON).....	35
4.6.	Instalador de la aplicación.	37
4.6.1.	Instalación de la aplicación.	38
5.	Repositorio del proyecto.	40
5.1.	Carpetas y archivos del repositorio.	40
5.1.1.	AppControlRobot (Aplicación e instalador).	40
5.1.2.	PS4controller.....	41
5.1.3.	PythonScripts.	41
Anexos.	42
Códigos de movimiento.	42
car_moveset_presentacion2024.py	42
Códigos de prueba para la librería XInput.	45
Códigos de prueba para la librería DirectInput.	47
Códigos de visión artificial.	48
CameraInterface_2.py	48

Códigos para el manejo del robot mediante Flask.	50
main_control_vs.py	50
vs_control_robot.py	51
vs_control_servo.py	54
Código de la aplicación.....	59

1. Robot ArmPi Pro de Hiwonder.

El robot ArmPi Pro es un robot desarrollado por la empresa Hiwonder, el centro de procesamiento del sistema se trata de una Raspberry Pi 4 modelo B, los códigos de control están desarrollados en base a ROS y utiliza Python para programar sus distintas directivas, módulos y algoritmos. El robot cuenta con un brazo robótico de cinco ejes con una pinza, una cámara USB por la cual se implementa un sistema de visión artificial, además cuenta con un chasis con ruedas mecánicas omnidireccionales controladas por medio de cuatro motores de corriente directa. Gracias a esto el robot puede realizar funciones de movimiento omnidireccional, agarre por medio de su pinza, seguimiento de objetos gracias a su sistema de visión, entre otras posibilidades.

El robot ArmPi Pro fue diseñado como un juguete y una herramienta didáctica de aprendizaje sobre robótica dirigida a los niños, por lo que el potencial de desarrollo de este se ve mermado debido a las limitaciones en su diseño y programación impuestas por la compañía Hiwonder.

2. Sistema operativo Linux – Ubuntu 18.04.

El robot ArmPi Pro cuenta con una distribución de Linux personalizada, tratándose de una modificación hecha por Hiwonder del sistema Ubuntu 18.04.6 LTS Bionic Beaver.

```
./+00sssoo+/-.  
`:+ssssssssssssssssss+:`  
-+ssssssssssssssssssyyssst-  
.osssssssssssssssssdMMMMNyssso.  
/ssssssssssshdmmNNmmyNNMMhsssss/  
+ssssssssshmydMMMMMMNdddyssssssst+  
/ssssssshNNMMMyhhyyyhmNNMMhsssss/  
.sssssssdMMMMhssssssshNNMMdssssss.  
+ssssshhhyNNMMNysssssssssyNNMMYssssst+  
osssyNNMMNyMMhssssssssssshmmhssssso  
osssyNNMMNyMMhssssssssssshmmhssssso  
+ssssshhhyNNMMNysssssssssyNNMMYssssst+  
.sssssssdMMMMhssssssshNNMMdssssss.  
/ssssssshNNMMYhhyyyhdNNMMhsssss/  
+sssssssdmydMMMMMMNdddyssssssst+  
/ssssssssshdmmNNmmyNNMMhsssss/  
.osssssssssssssssssdMMMMNyssso.  
-+ssssssssssssssssyyssst-  
`:+ssssssssssssssssss+:`  
./+00sssoo+/-.
```

```
ubuntu@ubuntu  
-----  
OS: Ubuntu 18.04.6 LTS aarch64  
Host: Raspberry Pi 4 Model B Rev 1.5  
Kernel: 5.4.0-1068-raspi  
Uptime: 13 mins  
Packages: 2723  
Shell: bash 4.4.20  
Terminal: /dev/pts/0  
CPU: (4) @ 1.500GHz  
Memory: 1295MiB / 3791MiB
```

Figura 2.0. Información del sistema.

Este sistema operativo cuenta con drivers, códigos de ejemplo, programas, servicios y todos los elementos para realizar los tutoriales de control del robot diseñados por la compañía.

2.1. Conexión vía SSH con el sistema.

Para manipular de forma remota los códigos y archivos dentro del sistema operativo Linux se requiere conectar la computadora o dispositivo desde el cual controlaremos el robot a la red de Wifi que genera él mismo (HW-E82EC0F1). Una vez conectado a esta red se puede utilizar la terminal de PowerShell o CMD de Windows, o una aplicación como PuTTY para realizar la conexión.

- Por medio de CMD o PowerShell.

En el caso de utilizar la terminal de Windows primero que nada se tiene que comprobar que se tenga instalado SSH, para ello se debe ingresar el comando:

```
ssh -V
```

El cual nos indicará la versión de SSH instalada en el equipo por medio de un mensaje similar al siguiente.

OpenSSH_for_Windows_8.6p1, LibreSSL 3.4.3

En caso de no tener instalado SSH se puede realizar su instalación siguiendo la guía de Microsoft:

https://learn.microsoft.com/es-es/windows-server/administration/openssh/openssh_install_firstuse?tabs=powershell

Una vez comprobada la instalación de SSH en el dispositivo se puede realizar la conexión con el robot por medio del comando:

```
ssh ubuntu@192.168.149.1
```

Al presionar “Enter” se nos pedirá una contraseña, la cual no podrá ser visible mientras se esté escribiendo; es decir que al pulsar las teclas los caracteres no aparecerán en la pantalla, pero sí serán procesadas por el sistema. La contraseña del sistema es **“hiwonder”**, al terminar de escribirla se debe pulsar una vez más “Enter”.

- Por medio de PuTTY.

Al conectarse por medio de la aplicación PuTTY se deberá escribir la dirección IP (192.168.149.1) en el apartado “Host Name (or IP address)”, mientras que en el apartado de “Port” se debe introducir el número 22. Una vez introducidos ambos datos se puede pulsar el botón “Open” y accederá a la conexión con el robot; para determinar la conexión solicita los datos del usuario y la contraseña, los cuales son **“ubuntu”** y **“hiwonder”** respectivamente.

Nota: Al conectarse por primera vez por cualquiera de los dos métodos surgirá una advertencia, en el caso de la terminal de Windows solamente hay que escribir “yes” y pulsar “Enter”, mientras que en PuTTY se tiene que seleccionar el botón “Ok” a la ventana de emergencia.

2.2. Softwares de control desarrollados por Hiwonder.

2.2.1. ArmPi Pro PC.

ArmPi Pro PC es un software proporcionado por la propia Hiwonder alojado dentro de la RaspberryPi, dentro de él se pueden encontrar las herramientas necesarias para operar los servomotores del brazo robótico. Al entrar al programa nos encontraremos con la siguiente interfaz:



Figura 2.2.1.0. Interfaz del software en chino

El software se encuentra originalmente en chino, pero en la parte superior izquierda podemos encontrar la opción para cambiar el lenguaje a inglés.



Figura 2.2.1.1 Interfaz del software en inglés.

Podemos apreciar como el software se compone de 3 áreas principales: la imagen del brazo (1), una sección con símil a una tabla de datos (2) y en la parte inferior una serie de botones (3).



Figura 2.2.1.2 Disposición del software.

Si damos un vistazo a la parte del brazo, podemos encontrar los siguientes elementos:

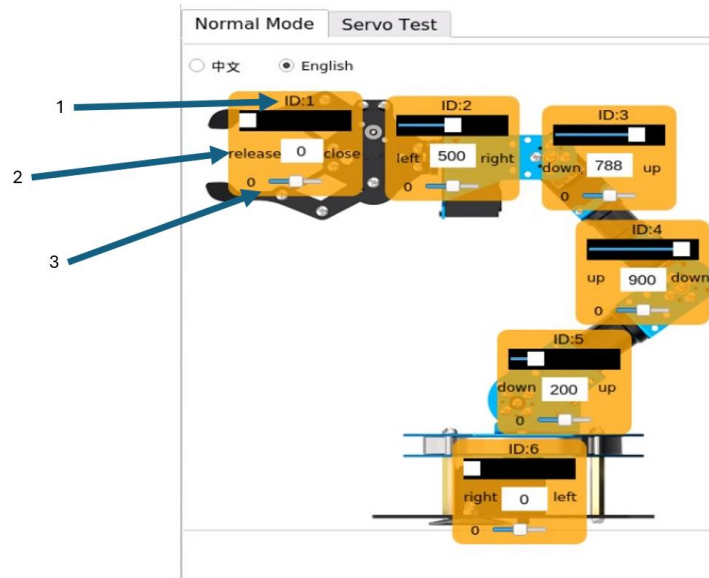


Figura 2.2.1.3 Disposición del brazo en el software.

1. Aquí podemos encontrar la ID del servomotor, que se representa del 1 al 6 empezando por la parte superior del brazo y terminando en la base de este.
2. Aquí se puede ajustar la posición del servomotor, los valores se encuentran en el rango de 0 a 1000.
3. También se puede ajustar la desviación de los servomotores. Los valores permitidos van desde -125 a 125.

En la siguiente sección encontramos el grupo de acciones que seguirá el robot.

Index	Time	ID:1	ID:2	ID:3	ID:4	ID:5	ID:6

Figura 2.2.1.4 Grupo de acciones del robot.

En el *index* se indica el número de acción. *Time* indica el tiempo de que le tomara a la acción realizarse. Por último, encontramos los valores de posición de los servomotores. En la parte de encima esta la ID del servomotor (1 al 6) y en la parte de abajo el valor (del 0 al 1000).

La sección dónde se encuentran los botones y otros iconos es bastante extensa por lo que únicamente se explicara de manera superficial la función de cada uno.

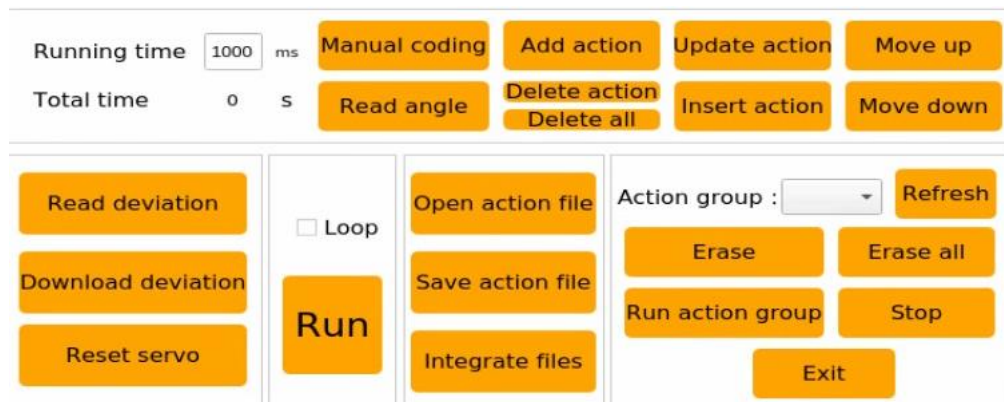


Figura 2.2.1.5 Iconos del software.

1. **Running Time** sirve para modificar el tiempo de ejecución de una simple acción.
2. **Total Time** es el tiempo total de ejecución del grupo de acciones.
3. **Manual coding** te permite modificar de manera manual la posición de los servomotores, eliminando la resistencia usual que se encuentra en los servomotores.
4. **Add action** añade los valores actuales de los servomotores a el grupo de acciones (esta se añade como la última acción).
5. **Delete accion** y **delete all** eliminan una de las acciones o todas las acciones que se encuentran en el grupo de acciones.
6. **Update action** toma la acción seleccionada del grupo de acciones y la reemplaza por los valores actuales de los servomotores (similar a add action).
7. **Insert action** inserta una acción después de la acción seleccionada.
8. **Move up** mueve la acción seleccionada una posición arriba en el grupo de acciones.

9. ***Move down*** mueve la acción seleccionada una posición abajo en el grupo de acciones.
10. ***Run*** ejecuta las acciones que se encuentran en el grupo de acciones una vez.
11. ***Loop*** permite correr el grupo de acciones en repetidas ocasiones.
12. ***Open action file*** permite abrir un grupo de acciones previamente guardado en el sistema.
13. ***Save action file*** permite guardar el grupo de acciones actual.
14. ***Integrate files*** te permite unir dos grupos de acciones. Para eso primero se debe de abrir un grupo de acciones con ***open action file***, y después se usa ***integrate files*** para unir todas las acciones.
15. ***Action group*** muestra el grupo de acciones.
16. ***Refresh*** recarga el grupo de acciones.
17. ***Erase*** elimina la acción seleccionada del grupo de acciones.
18. ***Precaución, Erase all*** elimina los archivos de grupos de acción que se encuentren en la carpeta del grupo de acciones abierto con ***open action file***.
19. ***Run action group*** ejecuta el grupo de acciones abierto con ***open action file*** una vez.
20. ***Stop*** para la ejecución del grupo de acción.
21. ***Exit*** se sale del software del ArmPi Pro.
22. ***Read deviation*** lee de manera automática la desviación guardada.
23. ***Download deviation*** descarga los valores de desviación que se configuraron en el software.
24. ***Rest Servo*** restaura la posición de los servomotores. La posición original corresponde al valor de 500.

2.2.2. LAB_Tool

Se tratar de un software interactivo mediante el cual el usuario puede experimentar con las funciones de reconocimiento de color por medio de la cámara USB de una forma muy básica. Este programa cuenta con pocas funciones y únicamente consiste en una serie de barras deslizantes mediante las cuales se ajusta la sensibilidad a los colores que se detectan.

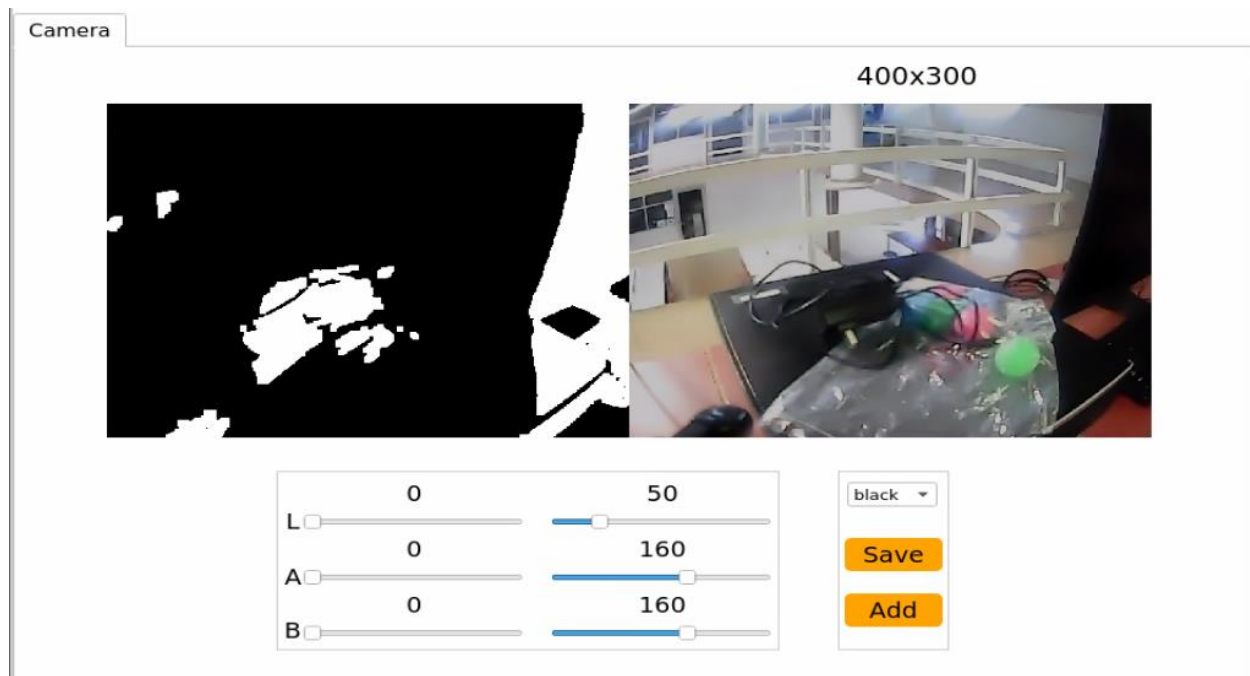


Figura 2.2.2.0. Interfaz de la aplicación LAB_Tool.

2.3. Actualización e instalación de paquetes del sistema operativo.

Para actualizar los paquetes y aplicaciones del sistema operativo de forma exitosa, es necesario conectar el robot a la red por medio de un cable ethernet. Esto es debido a que el robot Hiwonder genera su propia red Wifi, lo que le impide captar y conectarse a otras señales alrededor suyo.

Al realizar la conexión a la red se pueden ejecutar los siguientes comandos en terminal.

```
sudo apt update && sudo apt upgrade
```

De esta forma el sistema verifica si existen actualizaciones de los paquetes y aplicaciones descargadas, y los instala en su última versión.

2.3.1. Instalación de nuevos paquetes.

Buscando desarrollar los sistemas de control para el robot, se decidió instalar nuevas aplicaciones y paquetes de Python al sistema. Entre los más relevantes se encuentran los siguientes.

- Micro. Se trata de un editor de texto en terminal que nos permite crear, leer y modificar archivos de texto o código de forma sencilla sin la necesidad de una interfaz gráfica. A diferencia de otros editores de texto en terminal, Micro cuenta con un sistema de tabulación automática y coloreado de palabras, lo que lo convierte en una herramienta muy útil para la manipulación de código de forma remota.

Para instalarlo lo único que se requiere es escribir en terminal del sistema el siguiente comando:

```
curl https://getmic.ro | bash
```

- Flask. Consiste en un framework de desarrollo ligero y flexible para Python que permite crear aplicaciones web de forma rápida y sencilla. Por medio de este framework se pueden desarrollar páginas web que permiten el control de los periféricos de la Raspberry Pi y del sistema completo.

Para instalar Flask únicamente es necesario tener instalado Python y pip de forma previa, además de ejecutar el siguiente comando dentro de un entorno de Python en terminal:

```
pip install Flask
```

- V4L2-ctl. Video4Linux o V4L es una API de captura de video integrado con el núcleo Linux. Diversas webcams USB y otros periféricos son soportados. Por otra parte, V4L2-ctl es un paquete de Python que ofrece diversos comandos para el control de los drivers de V4L, permitiendo de forma sencilla la localización, identificación y manejo de los distintos periféricos de vídeo disponibles.

Para instalarlo también es necesario tener instalado Python y pip de forma previa, además de ejecutar el siguiente comando dentro de un entorno de Python en terminal:

```
pip install v4l2ctl
```

2.4. Inicialización de servicios desde el boot de Linux.

Con el objetivo de que el robot se encuentre en condiciones de funcionamiento en todo momento se requiere que sus drivers y scripts de control sean inicializados al energizar el robot. Para poder inicializar

scripts de esta forma se puede utilizar la herramienta de servicios de Systemd y otras herramientas que ofrece el sistema Linux.

Para que los controladores puedan ejecutarse durante el arranque o *'boot'* del sistema Linux es necesario utilizar un script de *'bash'*. Bash es un *'shell'* de Unix, es decir, una interfaz de línea de comandos cuyo fin es el de interactuar directamente con el sistema operativo. Este es el shell predeterminado en muchas distribuciones GNU/Linux, incluyendo Ubuntu. Su nombre es un acrónimo de Bourne Again SHell. Por convención se suele dar a este tipo de archivos la extensión *'sh'*.

Un archivo de *'bash'* es un archivo de texto sin formato que contiene una lista de comandos. Estos comandos son una combinación ordenada de los comandos que se desean ejecutar, escritos exactamente de la misma forma en la que se escribirían si se hiciera en una terminal. Cualquier comando que pueda ser normalmente ejecutado en la terminal puede ser listado en un archivo *'bash'* y realizará exactamente la misma acción.

Para crear el archivo *'bash'* lo primero que se necesita es crear un archivo con la extensión *'sh'* por medio de cualquier editor de texto que proporcione el sistema o se haya instalado en el mismo, los editores disponibles en el sistema del robot son nano, vim y micro. Los comandos listados para la creación de la mayoría de los scripts de control son los siguientes:

```
#!/bin/bash
```

Esta directiva del interprete o *'shebang line'* permite que el sistema conozca cuál intérprete o *'shell'* utilizará para ejecutar el resto de los comandos, en este caso se especifica que se utilizará *'bash'*. La directiva del interprete comienza con un símbolo de etiqueta, numeral o coloquialmente conocido como “gato”, seguido de un signo de cierre de exclamación. Posterior a este signo se debe especificar la ruta al intérprete que se desea usar.

A continuación, en los siguientes renglones del archivo se listan todos los comandos que se desean ejecutar. Todos los scripts de control desarrollados por nosotros, como se verá más adelante, fueron

desarrollados en Python; por lo que para su ejecución por medio de *'bash'* es necesario primero dirigirse al directorio del script por medio de un comando *'cd'*, para posteriormente ejecutar dicho script por medio del comando *'python3'*.

```
cd /ruta/hacia/el/archivo/  
python3 /ruta/hacia/el/archivo/nombre_del_archivo.py
```

Luego de crear y guardar el contenido del archivo *'bash'*, es necesario otorgarle permisos de ejecución para todos los usuarios del sistema; para esto se utiliza el comando *'chmod'* que, haciendo uso del parámetro *'x'*, le otorga dichos permisos al archivo.

```
chmod +x archivo_bash.sh
```

Este es el primer paso para ejecutar un script durante el arranque del sistema, para el siguiente paso se tiene que utilizar otra herramienta de Linux llamada *'systemd'*. Systemd es un conjunto de componentes básicos para un sistema Linux, proporciona un administrador de sistemas y servicios que se ejecuta e inicia el resto del sistema, es gracias a esta herramienta que se pueden inicializar varios drivers para los periféricos de la Raspberry Pi y también es gracias a esto que se inicializan muchas funcionalidades del sistema desarrolladas por Hiwonder como la señal de Wifi propia del robot.

Systemd incluye utilidades para controlar la configuración básica del sistema como el nombre de host, la fecha, la configuración regional, mantener una lista de usuarios conectados y contenedores y máquinas virtuales en ejecución, cuentas del sistema, directorios y configuraciones de tiempo de ejecución, y servicios para administrar una red simple. configuración, sincronización horaria de la red, reenvío de registros y resolución de nombres.

Entonces, lo que se busca es convertir el archivo *'bash'* en un servicio más del sistema, para lo cual se comienza creando un archivo con extensión *'service'* en el directorio donde se alojan todos los servicios:

```
/etc/systemd/system/.
```

Para esta acción se requieren permisos de super usuario, por lo que se debe utilizar *'sudo'* al ejecutar el comando:


```
sudo nano /etc/systemd/system/servicio_bash.service
```

Dentro de este archivo se deben escribir los siguientes parámetros:

```
[Unit]
```

```
Description=Descripción del servicio
```

```
After=network.target
```

```
[Service]
```

```
Type=simple
```

```
Restart=always
```

```
RestartSec=5
```

```
ExecStart=/ruta/hacia/el/archivo/archive_bash.sh
```

```
[Install]
```

```
WantedBy=multi-user.target
```

En el apartado titulado como '*Unit*' se proporciona metadatos descriptivos sobre el servicio en cuestión.

- '*Description*': Es una descripción legible para los usuarios.
- '*After*': Especifica las dependencias que deben iniciarse antes de este servicio. En este caso, indica que el servicio debe iniciarse posterior al servicio de red (*network.target*).

En el apartado titulado como '*Service*' se define cómo debe ejecutarse el servicio.

- '*Type*': Define cómo se debe ejecutar el proceso del servicio. En este caso, simple indica que el proceso se inicia directamente y systemd no espera a que termine antes de considerarlo iniciado correctamente.
- '*Restart*': Este parámetro indica debe reiniciarse automáticamente si se interrumpe o termina inesperadamente. En este caso está configurado para reiniciarse siempre.

- *'RestartSec'*: Especifica el tiempo en segundos que se debe esperar antes de intentar reiniciar el servicio.
- *'ExecStart'*: Especifica la ruta del script para iniciar el servicio. En este caso, se proporciona la ruta al script *'bash'* que se desea ejecutar.

Por último, en el apartado titulado como *'Install'* se define cómo se instalará y habilitará el servicio en el sistema.

- *'WantedBy'*: Indica el objetivo que quiere este servicio. *'multi-user.target'* significa que se iniciará en niveles de ejecución que admiten múltiples usuarios.

Como paso final, se requiere crear el enlace entre el nuevo servicio y el controlador del sistema, habilitarlo e iniciarlo. Para esto se utilizan los siguientes comandos en una terminal:

```
sudo systemctl daemon-reload
sudo systemctl enable servicio_bash.service
sudo systemctl start servicio_bash.service
```

3. Control del ArmPi Pro por medio de Python.

3.1. Control del chasis de ruedas omnidireccionales.

El chasis de ruedas omnidireccionales cuenta con varios códigos de prueba que permiten al usuario experimentar con los movimientos que puede realizar el robot, estos códigos funcionan en base a Python y más específicamente al módulo de ROS (rospy); estos códigos pueden encontrarse en la carpeta */home/ubuntu/armpi_pro/src/armpi_pro_demo/chassis_control_demo/*. Las funciones de movimiento vienen predefinidas, sin embargo, sus parámetros pueden ser modificados para que el robot realice el movimiento deseado.

La función principal del movimiento de los motores es *'set_velocity.publish()'* y sus parámetros principales son: velocidad, grados o ángulo de giro y rotación en eje.

```
#set_velocity.publish(##,##)
#Parametros:
#Velocidad
#Rango -100 a 100
#Grados de giro
#0 Derecha
#90 Adelante
#180 Izquierda
#270 Reversa
#Rotacion en eje
#-2 a 2
#Valor negativo: rotacion antihoraria
#Valor positivo: rotacion horaria
```

Figura 1.3.0. Parámetros de la función set_velocity.publish().

En base a esta función y a los códigos de prueba del movimiento se logró desarrollar una secuencia de movimiento propia con la que se experimentó movimientos básicos de un solo sentido, rotaciones y una combinación de ambas para recorrer un cuadrado; este código tiene como nombre *"car_moveset_presentacion2024.py"*. El archivo se encuentra anexado al final de este documento.

3.2. Control de los servomotores del brazo robótico.

En los ejemplos dados por Hiwonder, el control de los servomotores se hace mediante la librería Board, situada en cada una de las carpetas en las que se hace uso de esta. Como review general, el archivo Board.py contiene algunas funciones que facilitan el manejo de la placa de expansión situada sobre la Raspberry, a excepción del control de los motores DC por medio de Bus.

Los ejemplos para el control de los servomotores los puedes encontrar en */home/ubuntu/armpi_pro/src/armpi_pro_demo/expansion_board_demo*.

La función principal para el control de los servototes es “*Board.setBusServoPulse(Parámetro1, Parámetro2, Parámetro3)*” (se puede eliminar el uso de Board si importas todo: `from Board import *`; o si importas la función directamente: `from Board import setBusServoPulse`).

- **Parámetro1:** Id del servomotor (1 al 6).
- **Parámetro2:** Posición a la que se desea llegar (0 a 1000).
- **Parámetro3:** El tiempo que va a tardar en llegar a la posición deseada (en milisegundos).

No se debe de confundir este parámetro con delay/sleep, aun es necesario agregar otra línea con sleep para que no se solapen las instrucciones subsiguientes.

3.3. Control de la cámara USB.

3.3.1. Módulo de cámara de ROS.

Uno de los servicios con los que cuenta el robot se trata de la transmisión de la cámara USB, la cual está configurada para transmitir su imagen a través de un servidor web alojado en el puerto 8080 de su IP. El procesamiento de las imágenes de la cámara está basado en ROS y se divide en un total de tres módulos: la imagen “cruda” o directa de la cámara, la imagen con procesamiento visual y la imagen de “*lab_config*”.

La imagen directa se trata de la imagen obtenida directamente de la cámara sin ningún procesamiento de por medio.

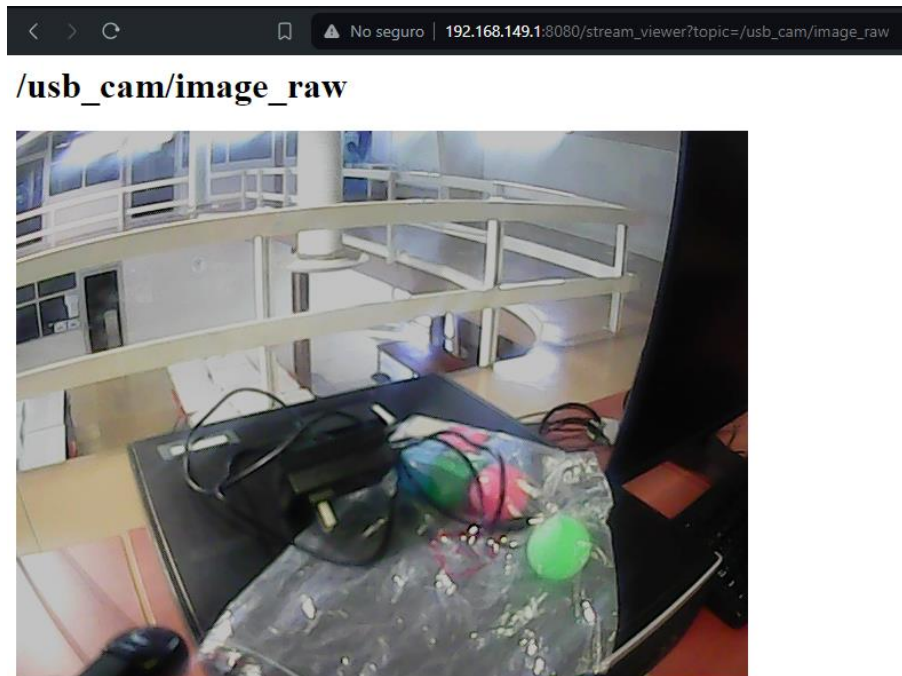


Figura 3.3.1.0. Cámara USB, imagen directa.

La imagen con procesamiento visual requiere de una activación previa de su módulo y su código en Python por medio de los siguientes comandos en terminal:

```
roslaunch object_tracking object_tracking_node.py
```

Se abre una terminal nueva sin cerrar la terminal anterior:

```
rosservice call /object_tracking/enter "{}"
```

Una vez activado este módulo se puede inicializar el seguimiento por colores (blue, red o green) y, en caso de que se desee, el seguimiento físico de dichos colores por medio del movimiento del robot.

```
rosservice call /color_tracking/set_target "data:  
'blue'"
```

En caso de que se desee apagar el módulo de procesamiento visual de la cámara se deberá ingresar el siguiente comando en terminal:

```
rosservice call /object_tracking/exit "{}"
```

Y pulsar la combinación de botones “Ctrl + C” en el primer terminal donde se ejecutó el archivo “*object_tracking_node.py*”.

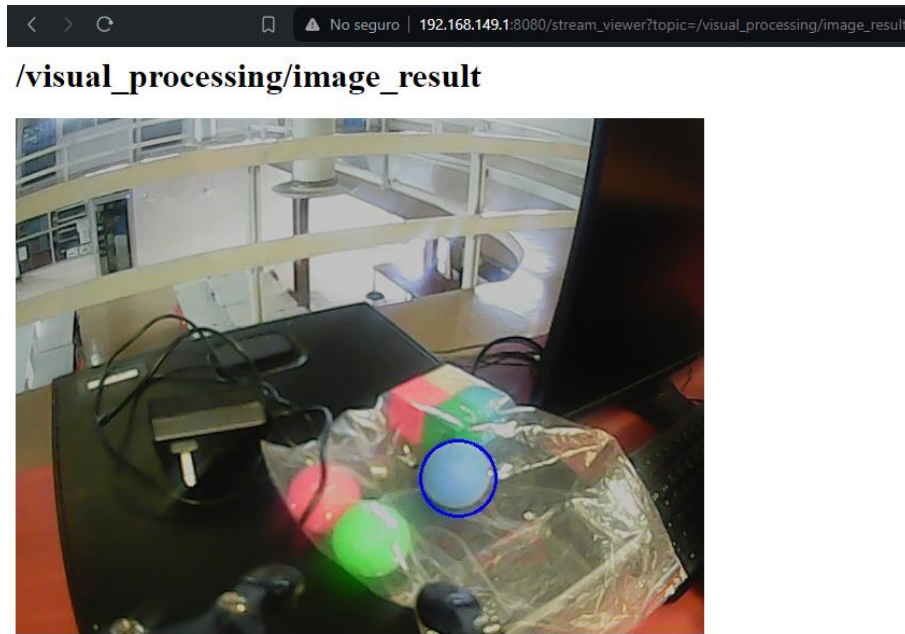


Figura 3.3.1.1. Cámara USB, imagen con procesamiento visual. Seguimiento del color azul.

La inicialización de estos módulos relacionados a la cámara se encuentra dentro de un archivo “start_node.sh” que los activa al inicializar el robot.

3.3.2. Transmisión y visión artificial con Flask y OpenCV.

Como se puede observar, la inicialización de los servicios de detección de colores que ofrece Hiwonder son toscos y poco amigables con el usuario, además de ser poco adaptables a una aplicación de escritorio debido al retraso entre la transmisión y lo mostrado en la aplicación. Para poder adaptar un servicio de transmisión de imagen más simple y fácil de manejar se optó por transmitir la imagen de la cámara USB directamente a una página web por medio de Flask y realizar la detección de colores aprovechando las capacidades de paquete OpenCV para Python.

El servidor web contiene una interfaz sencilla que consiste en la transmisión del vídeo como tal y cuatro botones que permiten seleccionar entre la detección de color verde, azul o rojo; o no detectar ningún color en lo absoluto.

Al integrar de forma conjunta la transmisión de vídeo con el sistema de visión, se puede controlar lo que el usuario desea ver mandando directamente las instrucciones al servidor, activando o desactivando la detección de colores sin la necesidad de ingresar a la terminal de Linux o inicializar otros nodos.

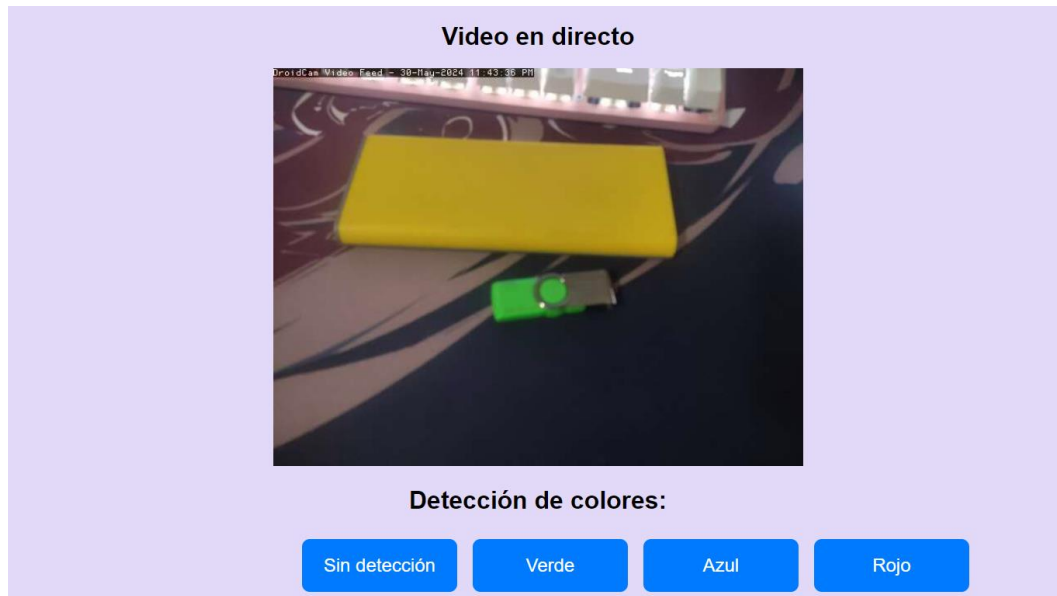


Figura 3.3.2.0. Transmisión de vídeo sin detección de colores.

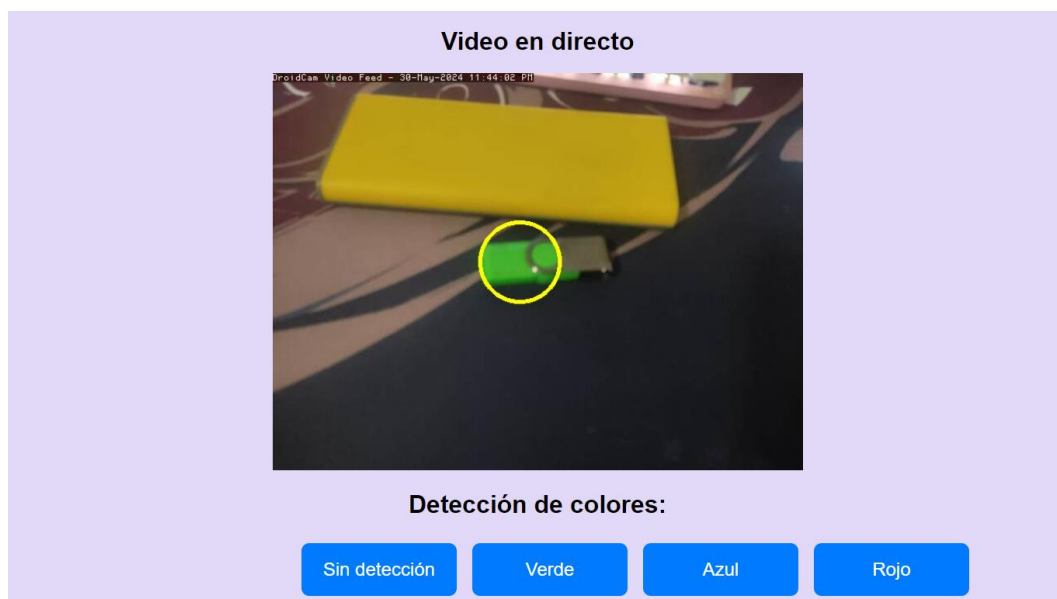


Figura 3.3.2.1. Transmisión de vídeo con detección de color verde.

El servidor web se inicia en el puerto 8081, justo uno después del puerto original de Hiwonder. De esta forma mantenemos un índice de servidores según su funcionalidad.

Este servidor de transmisión de imagen es uno de los servicios inicializados por “*Systemd*” en el arranque de Ubuntu, al inicializarlo después de activar los servicios de internet se logra que la transmisión de imagen ocurra desde el momento que el robot es encendido y se mantenga activa hasta que sea detenida de forma manual por el usuario o desarrollador.

4. Aplicación de control en Visual Studio 2019.

Para poder controlar de manera remota el robot se creó una aplicación de escritorio con el entorno de desarrollo Visual Studio 2019. A continuación, se describe la funcionalidad de la interfaz y del código.

Los recursos completos de la aplicación los puedes revisar en la siguiente liga de GitHub (actualmente en estado de desarrollo, por lo que se encuentra inaccesible de manera pública). Para más información consulte el apartado donde se habla acerca del repositorio del proyecto.

https://github.com/AlejandroMoHo/Remote_control_omnidirectional_robot

4.1. Interfaz gráfica y elementos básicos.

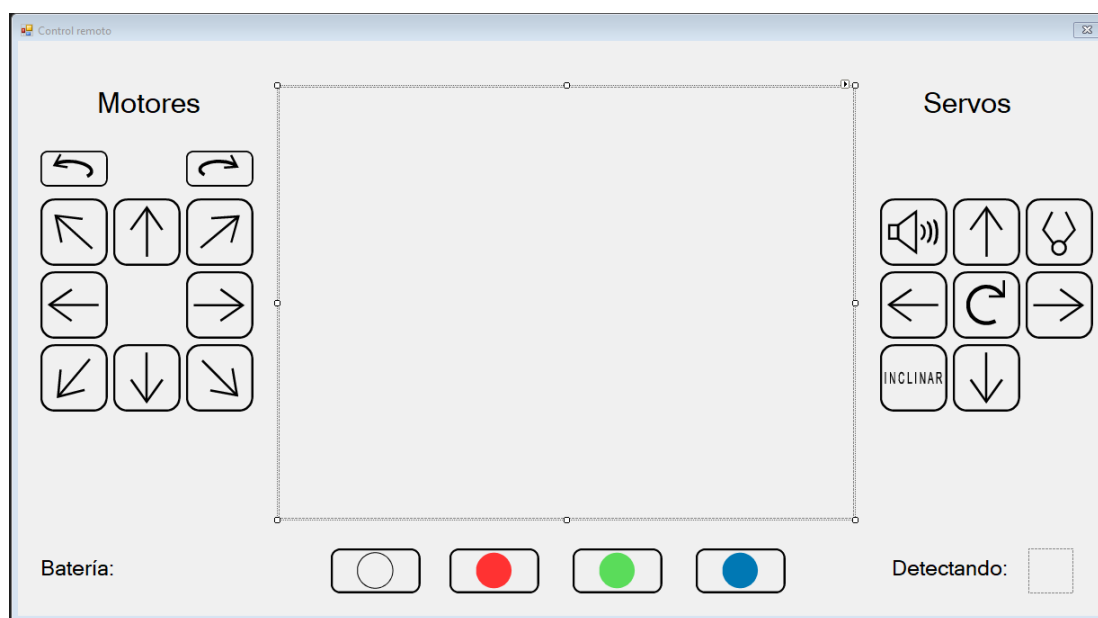


Figura 4.1.0. Interfaz gráfica creada con Visual Studio usando C#.

Se creó una interfaz que fuera lo más intuitiva posible. Del lado izquierdo se encuentran los botones para el control de los motores DC del robot. Cada flecha indica una de las direcciones programadas a las que se puede dirigir el robot, y las flechas superiores son para la rotación de este.

A la derecha encontramos los controles de los servomotores. En la versión actual solo se puede controlar directamente los servomotores responsables de subir y bajar la posición de la cámara (flechas

superior e inferior), así como la base del brazo (flechas laterales). También tiene funciones extra, como restaurar la posición de los servomotores (la posición “estándar” fue elegida por nosotros), abrir y cerrar la pinza, inclinar el brazo y activar un pequeño buzzer.

En el centro se encuentra el lugar donde se despliega la cámara.

Por último, en la parte inferior, tenemos un indicador porcentual de batería, el indicador es bastante inestable, pero en general da resultados aceptables que permiten darse una idea del porcentaje de batería del robot en ese momento. Además, tenemos unos botones que permiten alternar entre los colores que puede detectar la cámara, así como un indicador gráfico a la derecha del color seleccionado.

4.2. Conexión del control remoto e interpretación de los botones.

Como adición a la interfaz gráfica, también se puede conectar un control (de preferencia de Xbox, PS4 o PS5) a la computadora para darle instrucciones al robot. Para lo anterior se hizo uso de la librería “*XInput*” descargada directamente del administrador NuGet de Visual Studio.

4.2.1. Instalación de DS4 Windows.

En caso de que estes usando un control de Xbox esta sección no es necesaria, únicamente conecta el control a la computadora y abre la aplicación.

En caso de que estes usando un control de PS será necesario instalar DS4 Windows para mapear el control como si fuera uno de Xbox. Lo anterior se debe a que la librería *XInput* solo tiene compatibilidad con controles del último tipo, en anteriores versiones de la aplicación se hizo uso de *DirectInput* (el cual acepta controles de todo tipo), pero debido a ciertas limitaciones (no es capaz de detectar con precisión la posición el Joystick) se optó por sustituirlo.

Es muy sencillo instalar DS4 Windows, ya que realmente no estaremos instalando directamente el programa, sino que la página te descarga una versión portátil. La página es la siguiente:

<https://ds4-windows.com/download/ryochan7-ds4windows/>

Una vez descargado el archivo, descomprímelo, abre la carpeta DS4Windows y busca el ejecutable DS4Windows.exe. Si es tu primera vez usándolo te pedirá seguir una serie de pasos para instalar otras dependencias, solo llévalos a cabo y no debería de haber mayor complicación.

4.2.2. Instalación de la librería XInput.

Lo primero que debemos de hacer es crear un nuevo proyecto. Ahora debemos de dirigir nuestra atención al explorador de soluciones. En el podremos ver un archivo con el nombre de nuestro proyecto y un icono de C#.

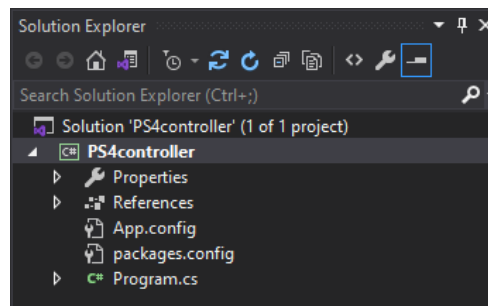


Figura 4.2.2.0. Explorador de soluciones de Visual Studio 2019.

Debemos de dar clic derecho en dicho archivo. Nos desplegará una serie de opciones, buscaremos la que dice “Manage NuGet Packages” y le daremos clic izquierdo.

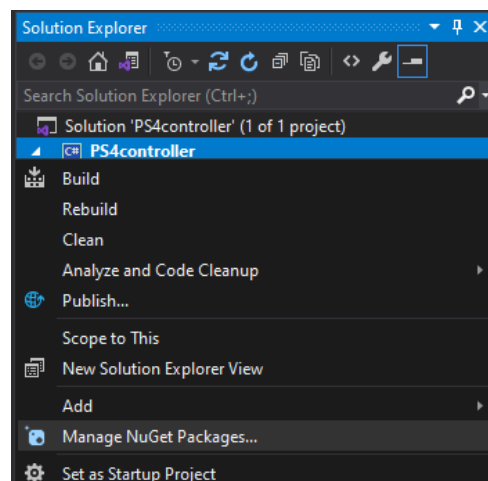


Figura 4.2.2.1. Menú de opciones de nuestro programa en Visual Studio 2019.

Nos abrirá una nueva pestaña donde podremos ver los paquetes NuGet que tenemos instalados, en nuestro caso, ya podemos ver cómo está instalado XInput.

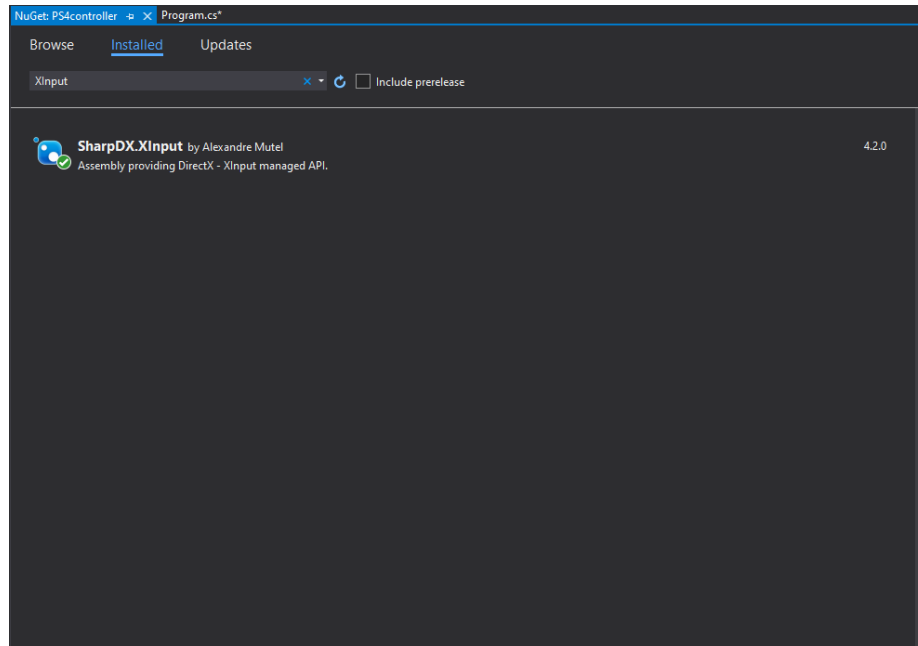


Figura 4.2.2.2. NuGet mánager de Visual Studio 2019.

Para descargar nuevos paquetes, solo es necesario ir a la pestaña “Browse” y en el buscador escribiremos “XInput”, la primera opción es el paquete que estamos buscando.

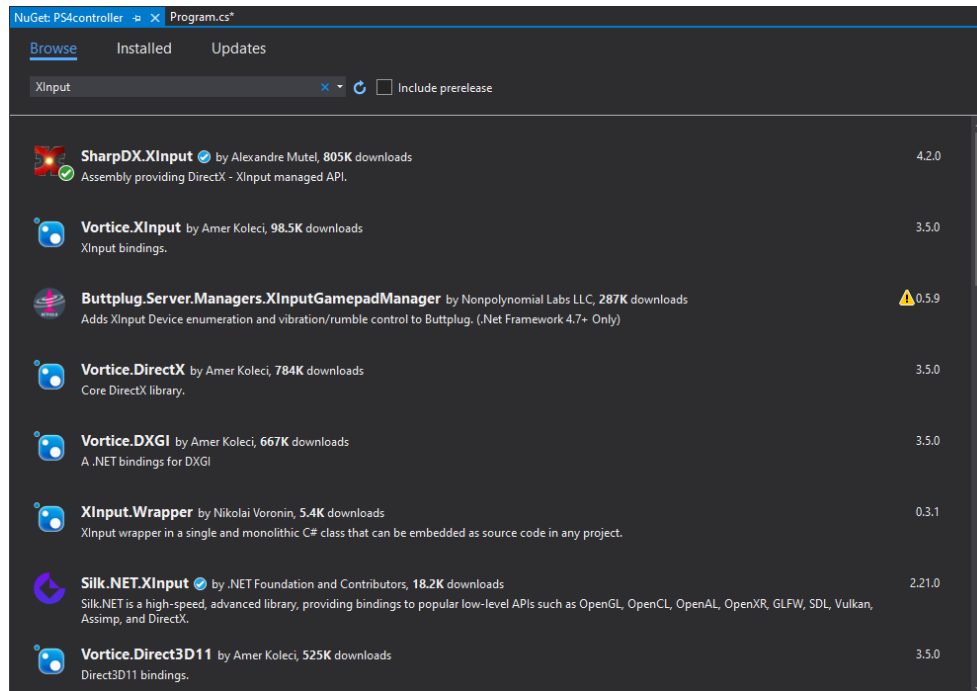


Figura 4.2.2.3. Explorador de paquetes de NuGet de Visual Studio 2019.

Al lado izquierdo debería de salirte un panel, donde podrás instalar, desinstalar, actualizar y ver la información del paquete.

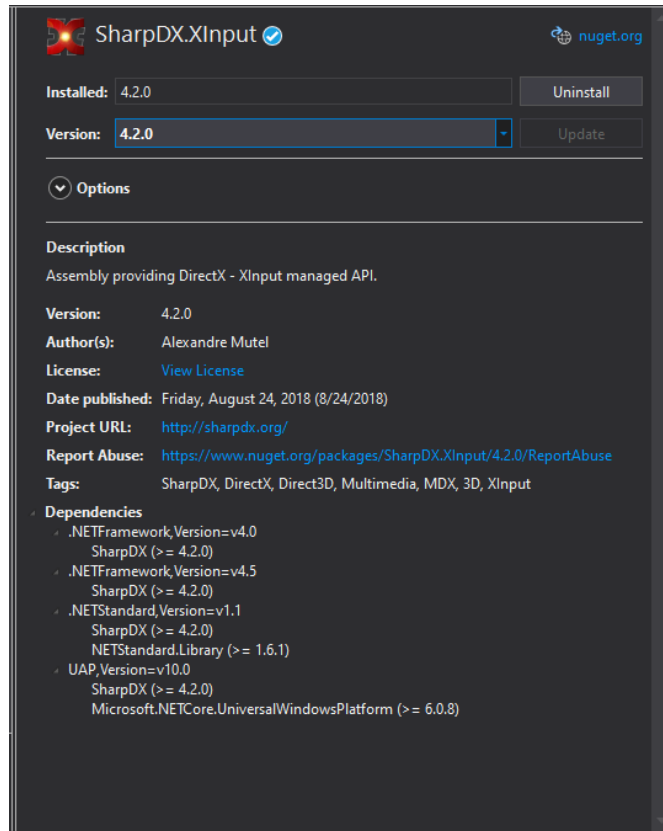


Figura 4.2.2.4. Opciones y características de los paquetes NuGet en Visual Studio 2019.

4.2.3. Uso de la librería XInput.

En caso de que no se tenga noción del uso de la librería, en la carpeta de “PS4controller” localizada en el git(o en la sección de anexos) se encuentra un programa muy sencillo que nos desplegara una consola que nos indicara los botones que se están presionando.

Nota: el programa tiene esta línea al final

```
Thread.Sleep(1000);
```

Por cuestiones de pruebas se dejó ese valor, pero si se desea que la respuesta desplegada en consola sea más responsiva se recomienda disminuir el tiempo de delay. Con este tiempo de delay se debe dejar presionado más de un segundo el botón deseado para que se despliegue en la consola.

4.3. Comunicación entre aplicación y los servidores.

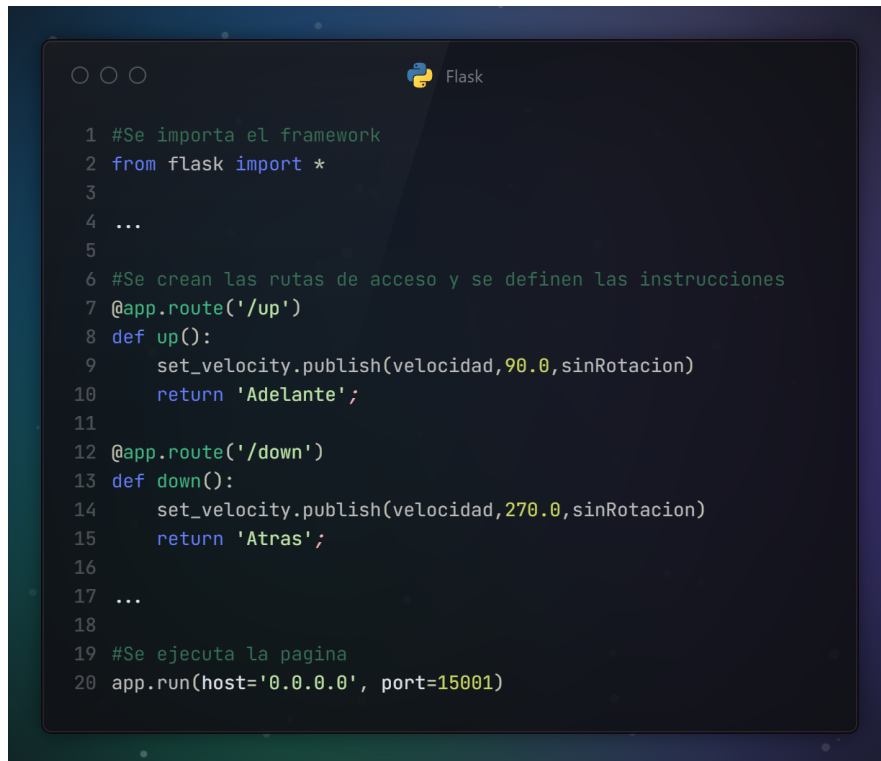
Para comunicar la aplicación con la Raspberry se optó por usar Flask. La idea es crear una página web, en donde por medio de direcciones se ejecutarán instrucciones en Python desde la Raspberry.

Ya que explicar todo el código podría ser repetitivo y engorroso solo se explicará cómo funciona el control de los motores, las demás funciones siguen una estructura similar.

4.3.1. Python (Flask).

Lo primero es crear un programa de Python para crear nuestra página web. La estructura básica es:

- Importar el framework.
- Declarar las rutas que tendrá la página web y definir la función que se realizará al momento de acceder a ellas.
- Se ejecuta el servidor web.

A screenshot of a code editor window titled 'Flask' showing a Python script. The script imports Flask, defines two routes ('/up' and '/down'), and runs the application on host '0.0.0.0' and port '15001'. The code is as follows:

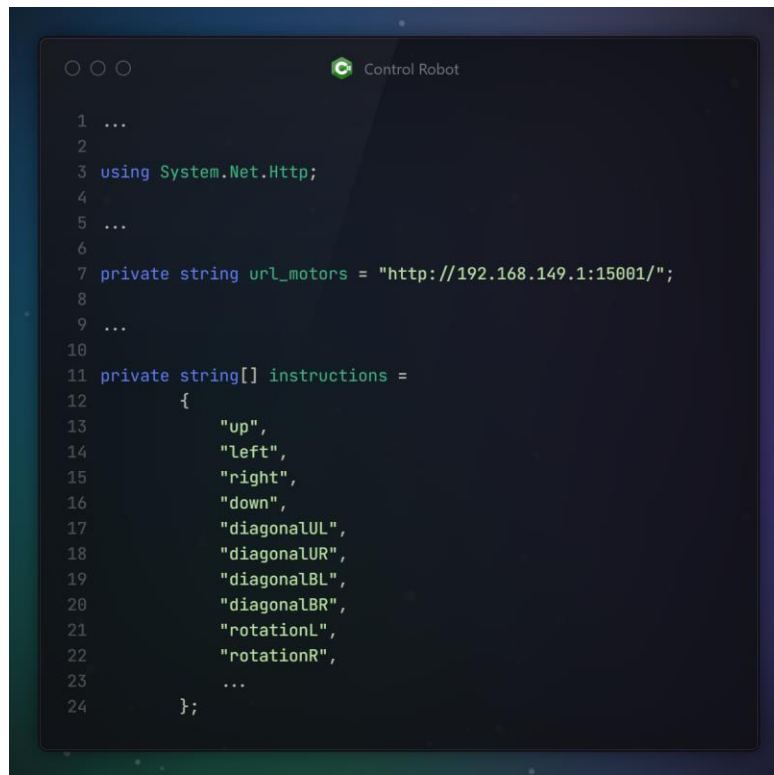
```
1 #Se importa el framework
2 from flask import *
3
4 ...
5
6 #Se crean las rutas de acceso y se definen las instrucciones
7 @app.route('/up')
8 def up():
9     set_velocity.publish(velocidad,90.0,sinRotacion)
10    return 'Adelante';
11
12 @app.route('/down')
13 def down():
14     set_velocity.publish(velocidad,270.0,sinRotacion)
15    return 'Atras';
16
17 ...
18
19 #Se ejecuta la pagina
20 app.run(host='0.0.0.0', port=15001)
```

Figura 4.3.1.0 Programa reducido sobre el uso de Flask en Python.

4.3.2. Visual Studio

Mediante Visual Studio se accede a las rutas de la página web creada con Flask. Para ello anteriormente se hizo uso de la herramienta “WebBrowser” que ofrece Visual Studio, pero esto hacía que fuera necesario cargar la página de manera recurrentemente. Debido a la anterior se infirió que sería mejor hacer llamadas directas a la API de la página web, haciendo uso de la librería (integrada por defecto) “System.Net.Http”.

Por cuestiones de comodidad, se decidió establecer una variable que contendrá la URL de la página, así como un arreglo donde estarán las instrucciones que se pueden ejecutar.



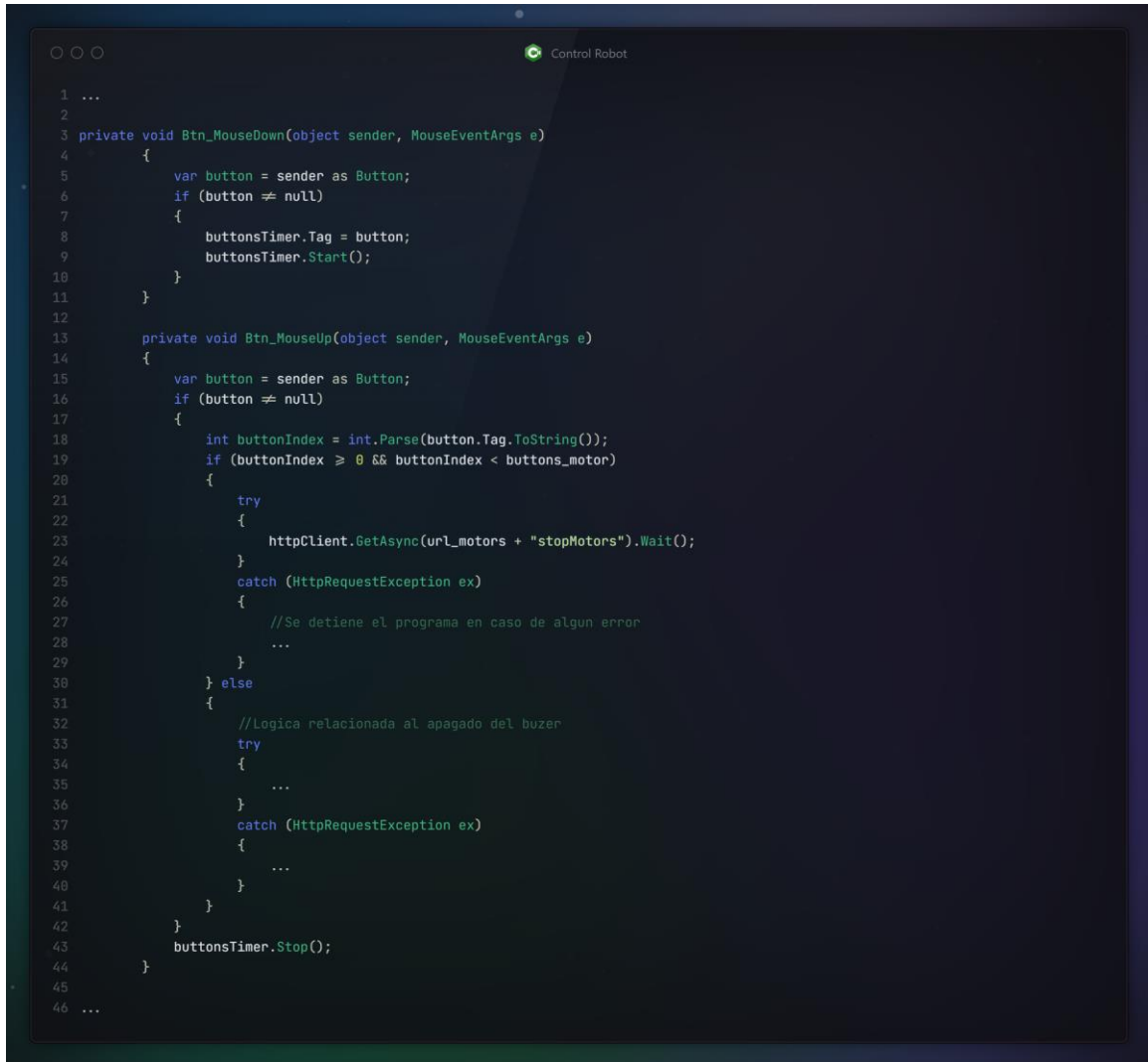
```
1 ...
2
3 using System.Net.Http;
4
5 ...
6
7 private string url_motors = "http://192.168.149.1:15001/";
8
9 ...
10
11 private string[] instructions =
12 {
13     "up",
14     "left",
15     "right",
16     "down",
17     "diagonalUL",
18     "diagonalUR",
19     "diagonalBL",
20     "diagonalBR",
21     "rotationL",
22     "rotationR",
23     ...
24 };
```

Figura 4.3.2.0. Librería, URL de la página que controla los motores y arreglo de las instrucciones.

Lo que sigue es crear nuestro cliente que nos permitirá hacer llamadas a la página web.

```
private HttpClient httpClient = new HttpClient();
```

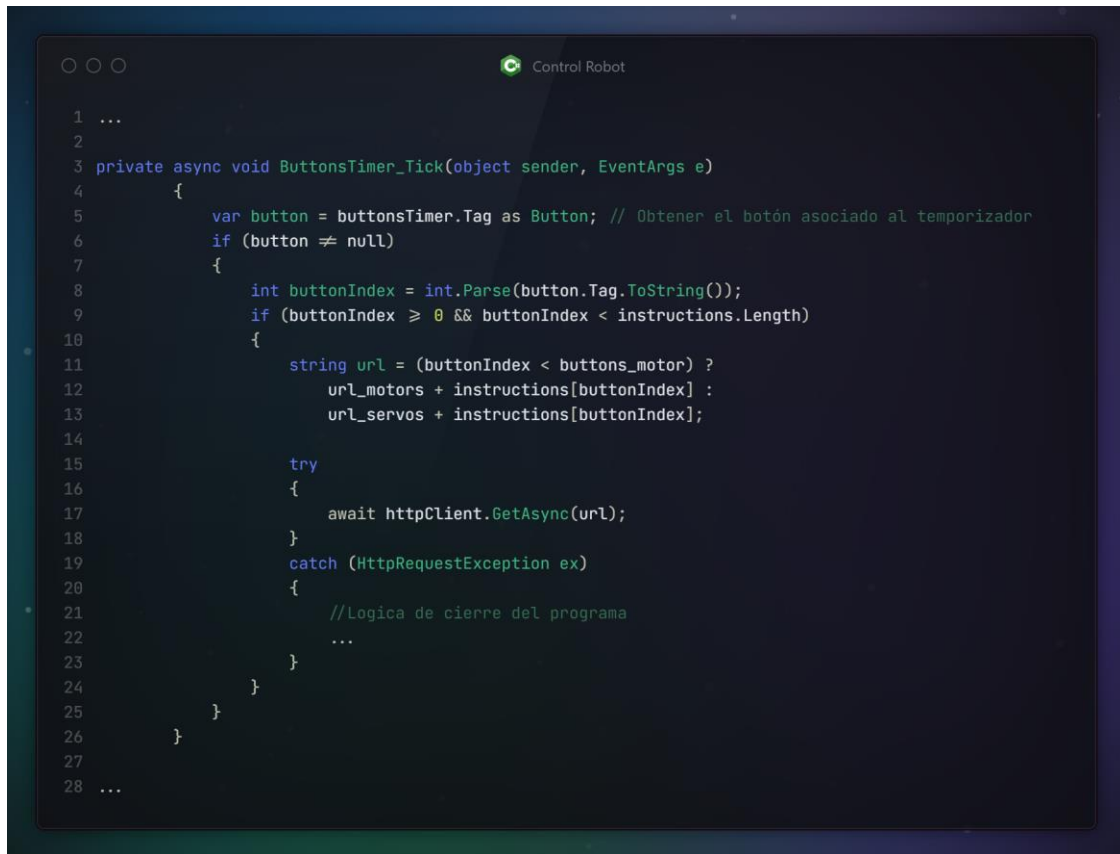

En el caso de los botones se tienen dos interacciones principales que todos los botones comparte: “MouseDown” y “MouseUp”; esto nos sirve para detectar cuando un botón es presionado y cuando se deja de presionar. En la primera interacción se hace revisión del valor de Tag relacionado al mismo y se almacena ese dato; y en la segunda, si alguno de los botones presionados está relacionado con el control de los motores se envía una señal para pararlos.



```
1 ...
2
3 private void Btn_MouseDown(object sender, MouseEventArgs e)
4 {
5     var button = sender as Button;
6     if (button != null)
7     {
8         buttonsTimer.Tag = button;
9         buttonsTimer.Start();
10    }
11 }
12
13 private void Btn_MouseUp(object sender, MouseEventArgs e)
14 {
15     var button = sender as Button;
16     if (button != null)
17     {
18         int buttonIndex = int.Parse(button.Tag.ToString());
19         if (buttonIndex >= 0 && buttonIndex < buttons_motor)
20         {
21             try
22             {
23                 httpClient.GetAsync(url_motors + "stopMotors").Wait();
24             }
25             catch (HttpRequestException ex)
26             {
27                 //Se detiene el programa en caso de algun error
28                 ...
29             }
30         } else
31         {
32             //Logica relacionada al apagado del buzzer
33             try
34             {
35                 ...
36             }
37             catch (HttpRequestException ex)
38             {
39                 ...
40             }
41         }
42     }
43     buttonsTimer.Stop();
44 }
45
46 ...
```

Figura 4.3.2.1. Código reducido de la detección de estados de los botones.

Lo último que nos queda es darle una instrucción al cliente según el botón que se presione.



```
1 ...
2
3 private async void ButtonsTimer_Tick(object sender, EventArgs e)
4 {
5     var button = buttonsTimer.Tag as Button; // Obtener el botón asociado al temporizador
6     if (button != null)
7     {
8         int buttonIndex = int.Parse(button.Tag.ToString());
9         if (buttonIndex >= 0 && buttonIndex < instructions.Length)
10        {
11            string url = (buttonIndex < buttons_motor) ?
12                url_motors + instructions[buttonIndex] :
13                url_servos + instructions[buttonIndex];
14
15            try
16            {
17                await httpClient.GetAsync(url);
18            }
19            catch (HttpRequestException ex)
20            {
21                //Logica de cierre del programa
22                ...
23            }
24        }
25    }
26 }
27
28 ...
```

Figura 4.3.2.1. Código reducido del uso del cliente para acceder a las rutas de la URL.

Podemos notar unos cuantos detalles interesantes:

- Para que el código funcione en su estado actual se hace uso de un timer, esto para que se ejecute de manera continua el comando si el usuario sigue presionando el botón; el timer tiene la propiedad async, esto tiene que ver con el uso del cliente, aunque no se profundizara en ello.
- Se revisa que haya presionado un botón, y en caso de ser así, se revisa la Tag que tenía este y se convierte en un índice para el arreglo de instrucciones.
- Según el Tag del botón se construye la URL a la que llamara el cliente. Se toma la URL donde está la página del control de los motores y a esto se le suma la instrucción correspondiente al botón.
- Se usa el cliente y se le hace la llamada a la URL construida anteriormente.

4.4. Conexión con el servidor de la cámara.

4.4.1. Configuración de la cámara usando qv4l2.

Es posible modificar algunos de los valores y características de la cámara. Este punto del reporte fue para probar la posibilidad del uso de la cámara en exteriores sin que se llegara a saturar, desafortunadamente no se tuvo éxito, aun así, se consideró que podría ser una buena referencia a futuro.

Para configurar la cámara es necesario instalar qv4l2 (aplicación de test bench para video4linux), para instalarlo solo es necesario escribir en la terminal:

```
sudo apt install qv4l2
```

Para ejecutarlo solo es necesario escribir el siguiente comando en la terminal:

```
qv4l2
```

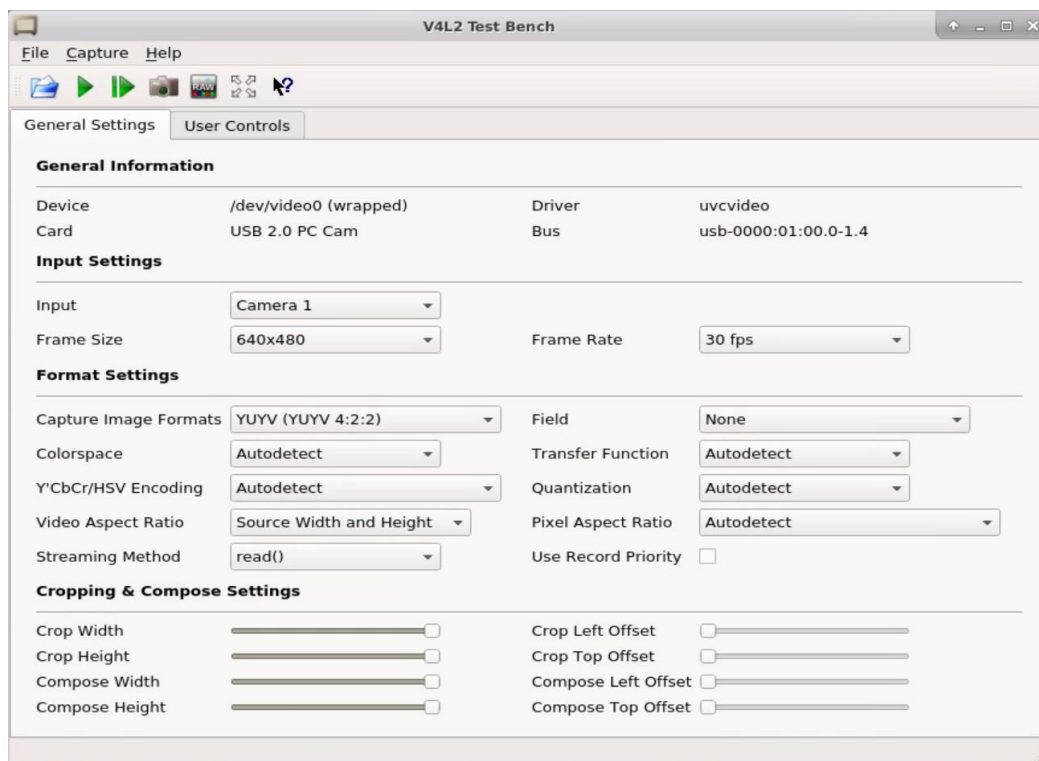


Figura 4.4.1.0. Ventana principal del Test Bench de V4L2.

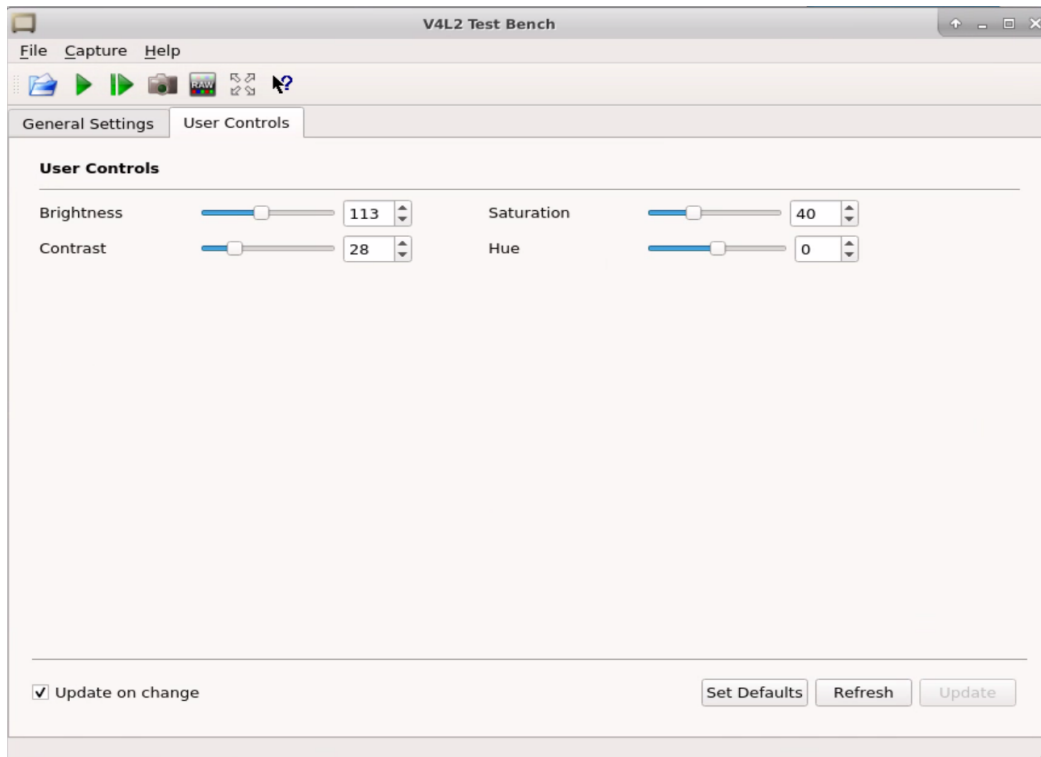


Figura 4.4.1.1. Ventana de controles del usuario del Test Bench de V4L2.

4.5. Obtención de los valores de batería (JSON).

El archivo Board.py ofrece una función para obtener el valor de carga de batería. Los datos leídos pueden resultar un poco confusos, pero después de realizar varias lecturas cuando la batería esta descargada y cargada totalmente, obtuvimos que los rangos leídos estaban entre los 6000 y los 8000.

Con eso en mente, ahora tenemos otro problema: ¿Cómo podemos pasar los datos de un Script de Python a nuestra aplicación de Visual Studio?

Afortunadamente Flask nos permite subir datos json a la página creada con este framework y es bastante sencillo.

1. Debemos de crear el arreglo que guardara la información de la función getBattery().

```
sensors = [  
    {
```

```

        "id":0,
        "type_sensor":"battery",
        "type_value":"int",
        "value":0
    }
]

```

2. Ahora debemos de crear una ruta para la página de flask.

```
@app.route('/get_sensors')
```

3. Por último, definimos la función y cargamos los datos en la página.

```

def get_battery():
    sensors[0]["value"] = Board.getBattery()
    return jsonify(sensors)

```

La función será llamada cada que se acceda a la ruta declarada.

Ahora solo queda recuperar los datos en Visual Code, y vamos a seguir una serie de pasos parecidos.

1. Incluir la librería “*Newtonsoft.Json*”.

```
using Newtonsoft.Json;
```

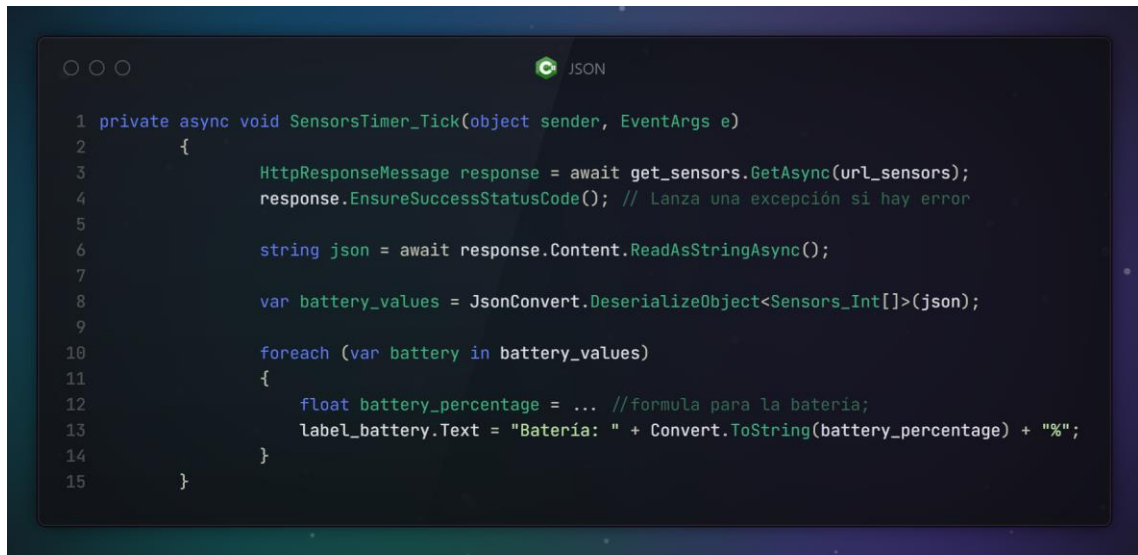
2. Necesitamos crear una clase donde desglosaremos nuestros datos.

```

public class Sensors_Int
{
    public int Id { get; set; }
    public string Type_Sensor { get; set; }
    public string Type_Value { get; set; }
    public int Value { get; set; }
}

```

3. Por último, debemos de realizar la función donde obtendremos los datos json.

A screenshot of a code editor window with a dark theme. The window has three window control buttons (minimize, maximize, close) in the top left and a tab labeled 'JSON' in the top center. The code is written in C# and is enclosed in a private async void method named SensorsTimer_Tick. The code performs an asynchronous GET request to a URL, checks the status code, reads the JSON response, deserializes it into an array of Sensors_Int objects, and then iterates over the array to update a label with the battery percentage. Line numbers 1 through 15 are visible on the left side of the code.

```
1 private async void SensorsTimer_Tick(object sender, EventArgs e)
2     {
3         HttpResponseMessage response = await get_sensors.GetAsync(url_sensors);
4         response.EnsureSuccessStatusCode(); // Lanza una excepción si hay error
5
6         string json = await response.Content.ReadAsStringAsync();
7
8         var battery_values = JsonConvert.DeserializeObject<Sensors_Int[]>(json);
9
10        foreach (var battery in battery_values)
11        {
12            float battery_percentage = ... //formula para la bateria;
13            label_battery.Text = "Bateria: " + Convert.ToString(battery_percentage) + "%";
14        }
15    }
```

Figura 4.5.0. Simplificación del código de obtención de los datos JSON.

Nota: el valor de “url_sensor” corresponde a la dirección donde se está guardando los datos json, en nuestro caso: "http://192.168.149.1:15002/get_sensors".

Para ver más en detalle el programa, recomendamos encarecidamente que se revise el git.

4.6. Instalador de la aplicación.

Con el fin de que la aplicación sea utilizable por cualquier persona, se decidió crear un instalador. Para generarlo se utilizó la extensión de Visual Studio 2019 llamado “Microsoft Visual Studio Installer Projects” desarrollado por Microsoft. Esta extensión nos permite personalizar el instalador del programa, incluyendo los autores de la aplicación, el nombre de la aplicación, el nombre del instalador, el ícono del instalador, si deseamos que el instalador se actualice por medio de una base de datos en línea o si se desea actualizar manualmente; entre otras características.

Para más información, se recomienda consultar el siguiente video tutorial:

<https://www.youtube.com/watch?v=c2NmtvENu3s>

Nosotros elegimos configurar el instalador con sus parámetros base, únicamente modificando nombres y autores.

4.6.1. Instalación de la aplicación.

El proceso de instalación comienza entrando al repositorio del proyecto, para encontrar la carpeta donde se aloja el instalador se recomienda consultar el apartado 5.1 donde se explica la distribución de las carpetas.

La ruta de la carpeta desde el inicio del repositorio es:

AppControlRobot/OmniRobot_Controller_Installer/Debug
https://github.com/AlejandroMoHo/Remote_control_omnidirectional_robot/tree/main/AppControlRobot/OmniRobot_Controller_Installer/Debug

Dentro de esta carpeta se pueden encontrar dos archivos: “OmniRobot_Controller_Installer.msi” y “setup.exe”, ambos archivos son necesarios para la instalación del programa, por lo que se recomienda descargar ambos. Al descargar los instaladores se pueden reconocer por su ícono y nombre.

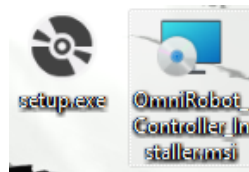


Figura 4.6.1.0. Ícono del instalador.

Al abrir el instalador se mostrará el asistente de instalación de la aplicación.

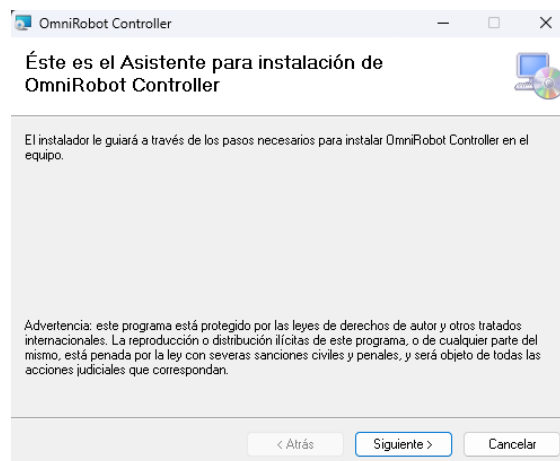


Figura 4.6.1.1. Asistente de instalación.

Al pulsar “Siguiente” se pedirá seleccionar la carpeta de instalación y si se pregunta si se desea instalar para todos los usuarios o solo para el actual.

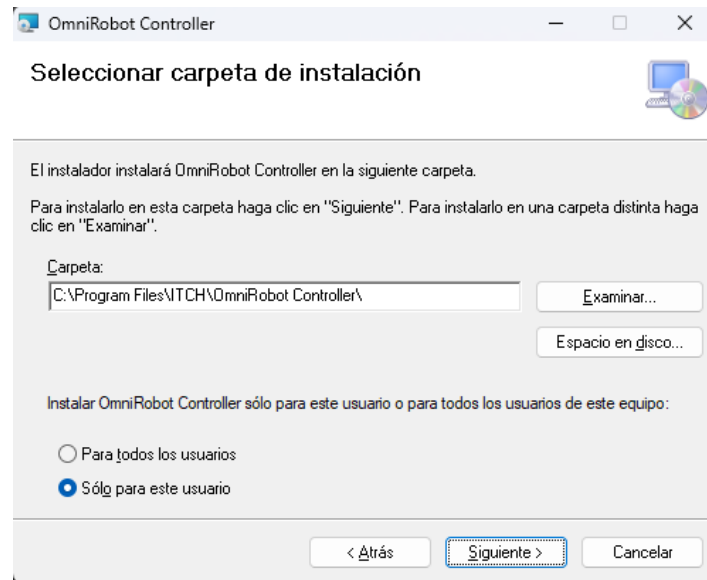


Figura 4.6.1.2. Ruta de instalación.

Posterior a esta pantalla, se tiene que pulsar “Siguiente” hasta que el programa se haya instalado y el asistente nos informe que el proceso se ha finalizado, se puede pulsar el botón de “Cerrar”.

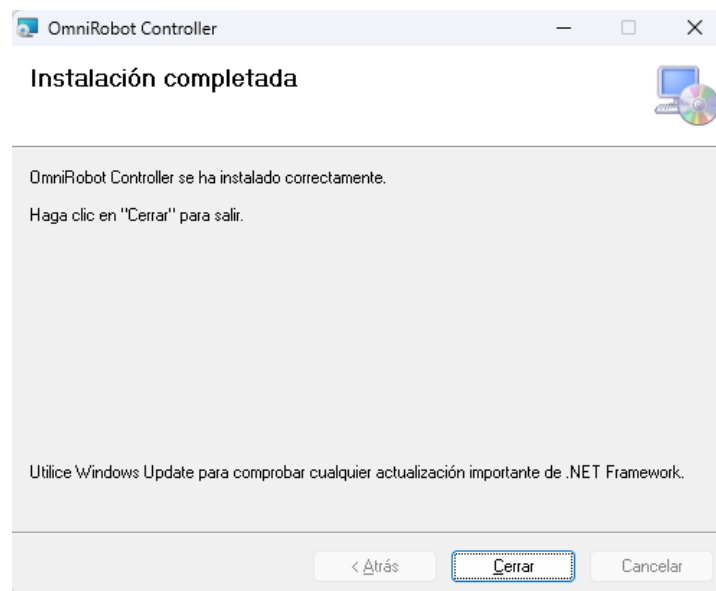


Figura 4.6.1.3. Instalación completada.

Para comprobar que el programa fue instalado correctamente se puede buscar su ícono en el escritorio o abrirlo directamente. **Nota:** Recuerde conectar un control antes de abrir la aplicación.

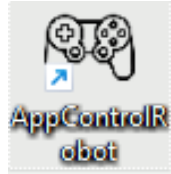


Figura 4.6.1.4. Ícono de la aplicación.

5. Repositorio del proyecto.

El proyecto de Visual Studio de la aplicación, al igual que los códigos de Python implementados dentro del robot, se encuentran alojados dentro de un repositorio público en la página web GitHub. El enlace de dicho repositorio es el siguiente.

https://github.com/AlejandroMoHo/Remote_control_omnidirectional_robot

Dentro de la carpeta raíz del repositorio se encuentran el manual completo del robot, el cual se encuentra leyendo en este momento.

5.1. Carpetas y archivos del repositorio.

El repositorio, dentro de su carpeta raíz, cuenta con tres carpetas principales dentro de las cuales se guardan los códigos y proyectos enfocados en el control de los motores y la cámara del robot.

5.1.1. AppControlRobot (Aplicación e instalador).

Dentro de esta carpeta se encuentran todas las carpetas relacionadas al proyecto de Visual Studio, entre las cuales se destacan dos principales:

- **AppControlRobot.** Dentro de esta carpeta se encuentran los códigos y recursos gráficos de la interfaz gráfica de control.

- **OmniRobot_Controller_Installer.** En el interior de esta carpeta se encuentra otra titulada “Debug”, dentro de la cual se encuentra el Setup o instalador del proyecto, disponible para su descarga.

5.1.2. PS4controller.

Se trata de una aplicación desarrollada en Visual Studio, la cual tiene como función principal la verificación del correcto funcionamiento del control, en caso de que se haya conectado un control de PS4 se recomienda utilizar esta aplicación.

5.1.3. PythonScripts.

En el interior de esta carpeta se encuentran los códigos principales implementados dentro de Ubuntu en la RaspberryPi, estos códigos son:

- CameraInterface_2.py: Es el código del sistema de detección de colores y su servidor dedicado.
- main_control_vs.py: Es el código del servidor que recibe todas las instrucciones para el manejo de los motores y servomotores del robot.
- vs_control_robot.py: Este código recibe instrucciones desde el main y se encarga del manejo de los motores de DC de las ruedas.
- vs_control_servo.py: Al igual que el anterior, este recibe instrucciones del main y se encarga del control de los servomotores del brazo robótico.

Anexos.

Códigos de movimiento.

car_moveset_presentacion2024.py

```
#Programa de rutina. Presentacion Robotica 2024

#Funcion:
#set_velocity.publish(##,##)
#Parametros:
#Velocidad
#Rango -100 a 100
#Grados de giro
#0 Derecha
#90 Adelante
#180 Izquierda
#270 Reversa
#Rotacion en eje
#-2 a 2
#Valor negativo: rotacion antihoraria
#Valor positivo: rotacion horaria

import sys
import rospy
from chassis_control.msg import *

if sys.version_info.major == 2:
    print('Ejecuta este programa con python3!')
    sys.exit(0)

print('Rutina Robotica 2024')

start = True

def stop():
    global start

    start = False
    print('Programa terminado...')
    set_velocity.publish(0,0,0) #Se paran los motores

if __name__ == '__main__':

    velocidad = 70 #-100 a 100
    velRotacion = 0.9 #-2 a 2
    delay = 1.3 #Segundos
```

```

#constantes
adelante = 90
atras = 270
derecha = 0
izquierda = 180
sinRotacion = 0

rospy.init_node('car_moveset_v1', log_level=rospy.DEBUG)
rospy.on_shutdown(stop)

set_velocity = rospy.Publisher('/chassis_control/set_velocity', SetVelocity, queue_size=1)

rospy.sleep(delay)
#####Rutina de prueba#####

##Adelante##
set_velocity.publish(velocidad,adelante,sinRotacion)
rospy.sleep(delay*6)
##Atras##
set_velocity.publish(velocidad,atras,sinRotacion)
rospy.sleep(delay*6)
##Derecha##
set_velocity.publish(velocidad,derecha,sinRotacion)
rospy.sleep(delay*6)
##Izquierda##
set_velocity.publish(velocidad,izquierda,sinRotacion)
rospy.sleep(delay*6)
##Rotacion horaria##
set_velocity.publish(velocidad,45,sinRotacion)
rospy.sleep(delay*5)
##Rotacion antihoraria##
set_velocity.publish(velocidad,135,sinRotacion)
rospy.sleep(delay*6)

##Cuadrado##
set_velocity.publish(velocidad,derecha,sinRotacion)
rospy.sleep(delay*6)
set_velocity.publish(velocidad,adelante,velRotacion)
rospy.sleep(delay)
set_velocity.publish(velocidad,derecha,sinRotacion)
rospy.sleep(delay*6)
set_velocity.publish(velocidad,adelante,velRotacion)
rospy.sleep(delay)
set_velocity.publish(velocidad,derecha,sinRotacion)
rospy.sleep(delay*6)
set_velocity.publish(velocidad,adelante,velRotacion)
rospy.sleep(delay)
set_velocity.publish(velocidad,derecha,sinRotacion)
rospy.sleep(delay*6)
set_velocity.publish(velocidad,adelante,velRotacion)

```

```
rospy.sleep(delay)
```

```
set_velocity.publish(0,0,0)  
print('Fin')
```

Códigos de prueba para la librería XInput.

```
using System;
using System.Threading;
using SharpDX.XInput;

namespace XboxController
{
    class Program
    {
        static void Main(string[] args)
        {
            var controller = new Controller(UserIndex.One);

            if (!controller.IsConnected)
            {
                Console.WriteLine("No hay control conectado =(");
                Environment.Exit(1);
            }

            Console.WriteLine("El control está conectado.");

            while (true)
            {
                var state = controller.GetState();
                var buttons = state.Gamepad.Buttons;

                // Verifica y muestra los botones específicos presionados
                if (buttons != GamepadButtonFlags.None)
                {
                    Console.WriteLine("Buttons pressed: " + buttons);

                    if ((buttons & GamepadButtonFlags.A) != 0)
                        Console.WriteLine("Button A pressed");
                    if ((buttons & GamepadButtonFlags.B) != 0)
                        Console.WriteLine("Button B pressed");
                    if ((buttons & GamepadButtonFlags.X) != 0)
                        Console.WriteLine("Button X pressed");
                    if ((buttons & GamepadButtonFlags.Y) != 0)
                        Console.WriteLine("Button Y pressed");
                    if ((buttons & GamepadButtonFlags.LeftShoulder) != 0)
                        Console.WriteLine("Left Shoulder pressed");
                    if ((buttons & GamepadButtonFlags.RightShoulder) != 0)
                        Console.WriteLine("Right Shoulder pressed");
                    if ((buttons & GamepadButtonFlags.Back) != 0)
                        Console.WriteLine("Back button pressed");
                    if ((buttons & GamepadButtonFlags.Start) != 0)
                        Console.WriteLine("Start button pressed");
                    if ((buttons & GamepadButtonFlags.LeftThumb) != 0)
                        Console.WriteLine("Left Thumbstick button pressed");
                    if ((buttons & GamepadButtonFlags.RightThumb) != 0)
                        Console.WriteLine("Right Thumbstick button pressed");
                }

                // Muestra los estados de los thumbsticks solo si hay movimiento
                var leftThumbX = state.Gamepad.LeftThumbX;
                var leftThumbY = state.Gamepad.LeftThumbY;
            }
        }
    }
}
```

```

        var rightThumbX = state.Gamepad.RightThumbX;
        var rightThumbY = state.Gamepad.RightThumbY;

        if (leftThumbX != 0 || leftThumbY != 0)
            Console.WriteLine($"Left Thumbstick: X={leftThumbX},
Y={leftThumbY}");
        if (rightThumbX != 0 || rightThumbY != 0)
            Console.WriteLine($"Right Thumbstick: X={rightThumbX},
Y={rightThumbY}");

        // Muestra los estados de los triggers solo si se utilizan
        var leftTrigger = state.Gamepad.LeftTrigger;
        var rightTrigger = state.Gamepad.RightTrigger;

        if (leftTrigger > 0)
            Console.WriteLine($"Left Trigger: {leftTrigger}");
        if (rightTrigger > 0)
            Console.WriteLine($"Right Trigger: {rightTrigger}");

        Thread.Sleep(1000);
    }
}
}
}

```

Códigos de prueba para la librería DirectInput.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using SharpDX.DirectInput;
namespace PS4controller
{
    class Program
    {
        static void Main(string[] args)
        {
            var directInput = new DirectInput();

            var joystickGuid = Guid.Empty;

            foreach (var deviceInstance in directInput.GetDevices(
                DeviceType.Gamepad, DeviceEnumerationFlags.AllDevices
            ))
                joystickGuid = deviceInstance.InstanceGuid;

            if(joystickGuid == Guid.Empty)
                foreach (var deviceInstance in directInput.GetDevices(
                    DeviceType.Joystick, DeviceEnumerationFlags.AllDevices
                ))
                    joystickGuid = deviceInstance.InstanceGuid;

            if(joystickGuid == Guid.Empty)
            {
                Console.WriteLine("No hay control conectado =(");
                Environment.Exit(1);
            }

            var joystick = new Joystick(directInput, joystickGuid);

            Console.WriteLine("El control es {0}", joystickGuid);

            joystick.Properties.BufferSize = 128;

            joystick.Acquire();

            while (true)
            {
                joystick.Poll();
                var data = joystick.GetBufferedData();

                foreach (var state in data)
                {
                    //if(state.Offset.ToString().Contains("Buttons0"))
                    Console.WriteLine(state);
                }
            }
        }
    }
}
```


Códigos de visión artificial.

CameraInterface_2.py

```
from flask import *
import cv2
import numpy as np

lower_color = np.array([0, 0, 0], np.uint8)
upper_color = np.array([0, 0, 0], np.uint8)
lower_color2 = np.array([0, 0, 0], np.uint8)
upper_color2 = np.array([0, 0, 0], np.uint8)
contourColor = (0, 255, 255)

kernel = np.ones((3, 3), np.uint8)

app = Flask(__name__)

# Función para detectar objetos de un color específico en una imagen
def detectColor(image, lower_color, upper_color):
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    mask = cv2.inRange(hsv, lower_color, upper_color)
    mask = cv2.erode(mask, kernel, iterations = 1)
    mask = cv2.dilate(mask, kernel, iterations = 1)
    contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    return contours

def drawCircle(frame):
    color_contours = detectColor(frame, lower_color, upper_color)
    color_contours2 = detectColor(frame, lower_color2, upper_color2)
    color_contours_total = color_contours + color_contours2

    for c in color_contours_total:
        area = cv2.contourArea(c)
        if area > 2500:
            #cv2.drawContours(frame, [c], 0, contourColor, 2)
            x,y,w,h = cv2.boundingRect(c)
            M = cv2.moments(c)
            if (M["m00"]!=0): M["m00"] = 1
            xCenter = int(M["m10"]/M["m00"])
            yCenter = int(M["m01"]/M["m00"])
            radius = xCenter-x
            cv2.circle(frame, (xCenter, yCenter), radius, contourColor, 3)

    return

def generate():
    cap = cv2.VideoCapture(0)
    while True:
        ret, frame = cap.read()
        if not ret:
            break
        else:
            drawCircle(frame)

        suc, encode = cv2.imencode('.jpg', frame)
        frame = encode.tobytes()
```

```

        yield(b'--frame\r\n' b'content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')
    cap.release()

@app.route('/')
def index():
    return render_template("index_camera.html")

@app.route('/video_feed')
def video_feed():
    return Response(generate(),
                    mimetype = 'multipart/x-mixed-replace; boundary=frame')

# Sin detección de objetos
@app.route('/cameraRaw')
def cameraRaw():
    global lower_color, upper_color, lower_color2, upper_color2, contourColor
    lower_color = upper_color = lower_color2 = upper_color2 = np.array([0, 0, 0])
    return redirect(url_for('index'))

# Detección de objetos verdes
@app.route('/detectGreen')
def detectGreen():
    global lower_color, upper_color, lower_color2, upper_color2, contourColor
    lower_color = np.array([35, 100, 100],np.uint8)
    upper_color = np.array([90, 255, 255],np.uint8)
    lower_color2 = upper_color2 = np.array([0, 0, 0],np.uint8)
    return redirect(url_for('index'))

# Detección de objetos azules
@app.route('/detectBlue')
def detectBlue():
    global lower_color, upper_color, lower_color2, upper_color2, contourColor
    lower_color = np.array([100, 100, 100],np.uint8)
    upper_color = np.array([125, 255, 255],np.uint8)
    lower_color2 = upper_color2 = np.array([0, 0, 0],np.uint8)
    return redirect(url_for('index'))

# Detección de objetos rojos
@app.route('/detectRed')
def detectRed():
    global lower_color, upper_color, lower_color2, upper_color2, contourColor
    lower_color = np.array([0, 50, 50],np.uint8)
    upper_color = np.array([10, 255, 255],np.uint8)
    lower_color2 = np.array([150, 120, 120],np.uint8)
    upper_color2 = np.array([179, 255, 255],np.uint8)
    return redirect(url_for('index'))

@app.teardown_request
def teardown_request(exception):
    cv2.destroyAllWindows()

if __name__ == '__main__':
    app.run(host='0.0.0.0', port='10300', debug=True)

```

Códigos para el manejo del robot mediante Flask.

main_control_vs.py

```
import subprocess as subp
import threading
import signal
import sys

dir1 = r'/home/ubuntu/Desktop/chassis_control_demo/vs_control_robot.py'
dir2 = r'/home/ubuntu/arduino_pro/src/arduino_pro_demo/expansion_board_demo/vs_control_servo.py'

# Evento para indicar a los hilos que deben detenerse
stop_event = threading.Event()

def flask_motores():
    try:
        result = subp.run(['python3', dir1], check=True)
        print(f'flask_motores terminó con código {result.returncode}')
    except subp.CalledProcessError as e:
        print(f'Error en flask_motores: {e}', file=sys.stderr)
    except Exception as e:
        print(f'Excepción en flask_motores: {e}', file=sys.stderr)
    finally:
        stop_event.set()

def flask_servo_sudo():
    try:
        result = subp.run(['sudo', 'python3', dir2], check=True)
        print(f'flask_servo_sudo terminó con código {result.returncode}')
    except subp.CalledProcessError as e:
        print(f'Error en flask_servo_sudo: {e}', file=sys.stderr)
    except Exception as e:
        print(f'Excepción en flask_servo_sudo: {e}', file=sys.stderr)
```

```

        finally:
            stop_event.set()

# Función para manejar la señal SIGINT (Ctrl+C)
def signal_handler(sig, frame):
    print("Interrupción recibida, cerrando hilos...")
    stop_event.set() # Indica a los hilos que deben detenerse
    thread1.join()   # Espera a que el hilo 1 termine
    thread2.join()   # Espera a que el hilo 2 termine
    sys.exit(0)      # Cierra el programa

# Registrar el manejador de la señal SIGINT
signal.signal(signal.SIGINT, signal_handler)

# Crear y arrancar los hilos
thread1 = threading.Thread(target=flask_motores)
thread2 = threading.Thread(target=flask_servo_sudo)

thread1.start()
thread2.start()

# Esperar a que los hilos terminen
thread1.join()
thread2.join()

# Comprobar si algún hilo ha activado el evento de parada
if stop_event.is_set():
    print("Uno o ambos hilos han terminado con error o se ha alcanzado el final del código.")
else:
    print("Ambos hilos han terminado correctamente.")

vs_control_robot.py

print('Control de motores mediante VS y flask')

```

```

from flask import *
import sys
import rospy
from chassis_control.msg import *

if sys.version_info.major == 2:
    print('Ejecuta este programa con python3!')
    sys.exit(0)

app = Flask(__name__)

velocidad = 200 #-100 a 100
velRotacion = 0.7 # -2 a 2
delay = 1.3 #Segundos

#constantes
adelante = 90
atras = 270
derecha = 0
izquierda = 180
sinRotacion = 0

def stop():
    global start

    start = False
    print('Programa terminado...')
    set_velocity.publish(0,0,0) #Se paran los motores

rospy.init_node('vs_control_robot', log_level=rospy.DEBUG)
rospy.on_shutdown(stop)

set_velocity = rospy.Publisher('/chassis_control/set_velocity', SetVelocity,
queue_size=1)

```

```
start = True

@app.route('/up')
def up():
    set_velocity.publish(velocidad,90.0,sinRotacion)
    return 'Adelante';

@app.route('/down')
def down():
    set_velocity.publish(velocidad,270.0,sinRotacion)
    return 'Atras';

@app.route('/left')
def left():
    set_velocity.publish(velocidad,180.0,sinRotacion)
    return 'Izquierda';

@app.route('/right')
def right():
    set_velocity.publish(velocidad,0.0,sinRotacion)
    return 'Derecha';

@app.route('/diagonalUL')
def diagonalUL():
    set_velocity.publish(velocidad, 135.0, sinRotacion)
    return 'DiagonalSI'

@app.route('/diagonalUR')
def diagonalUR():
    set_velocity.publish(velocidad, 45.0, sinRotacion)
    return 'DiagonalSD'

@app.route('/diagonalBL')
def diagonalBL():
    set_velocity.publish(velocidad, 225.0, sinRotacion)
```

```

        return 'DiagonalII'

@app.route('/diagonalBR')
def diagonalBR():
    set_velocity.publish(velocidad, 315.0, sinRotacion)
    return 'DiagonalID'

@app.route('/rotationL')
def rotationL():
    set_velocity.publish(velocidad, 90.0, velRotacion)
    return 'DiagonalSI'

@app.route('/rotationR')
def rotationR():
    set_velocity.publish(velocidad, 90.0, -velRotacion)
    return 'DiagonalSI'

@app.route('/stopMotors')
def stopMotors():
    set_velocity.publish(0,0,0)
    return 'Parar'

app.run(host='0.0.0.0', port=15001)

```

vs_control_servo.py

```

print('Control de servo motores mediante VS y flask')

```

```

from flask import *
from sys import exit
from time import sleep
import Board

sensors = [
    {
        "id":0,

```

```

        "type_sensor": "battery",
        "type_value": "int",
        "value": 0
    }
]

delay_movimiento = 100 #Solo referente al movimineto con la cruceta
delay_rutina_agarrar = 1000
delay_rutina_inclinar = 1000
sensibilidad_movimiento = 10 #Usar preferentemente valores entre 10-100

pos_base = 500
pos_cabeza = 150

tomar = True
inclinar = True

app = Flask(__name__)

@app.route('/up')
def up():
    global pos_cabeza
    global sensibilidad_movimiento
    pos_cabeza = pos_cabeza + sensibilidad_movimiento
    if(pos_cabeza>1000):
        pos_cabeza = 1000
    Board.setBusServoPulse(3,pos_cabeza,delay_movimiento)
    sleep(delay_movimiento/1000)
    return 'Adelante'

@app.route('/down')
def down():
    global pos_cabeza

```



```

global sensibilidad_movimiento
if(pos_cabeza<0):
    pos_cabeza = 0
pos_cabeza = pos_cabeza - sensibilidad_movimiento
Board.setBusServoPulse(3,pos_cabeza,delay_movimiento)
sleep(delay_movimiento/1000)
return 'Atras'

@app.route('/left')
def left():
    global pos_base
    global sensibilidad_movimiento
    pos_base = pos_base + sensibilidad_movimiento
    if(pos_base>1000):
        pos_base = 1000
    Board.setBusServoPulse(6,pos_base,delay_movimiento)
    sleep(delay_movimiento/1000)
    return 'Izquierda'

@app.route('/right')
def right():
    global pos_base
    global sensibilidad_movimiento
    pos_base = pos_base - sensibilidad_movimiento
    if(pos_base<0):
        pos_base = 0
    Board.setBusServoPulse(6,pos_base,delay_movimiento)
    sleep(delay_movimiento/1000)
    return 'Derecha'

@app.route('/dance')
def dance():
    return 'baile'

```

```

@app.route('/buzzer_on')
def buzzer_on():
    Board.setBuzzer(1)
    return 'buzzer_on'

@app.route('/buzzer_off')
def buzzer_off():
    Board.setBuzzer(0)
    return 'buzzer_off'

@app.route('/pick_leave')
def pick_leave():
    global tomar
    if (tomar == True):
        print('tomar')
        Board.setBusServoPulse(1,800,delay_rutina_agarrar)
        sleep(delay_rutina_agarrar/1000)
        tomar = False
    else:
        print('dejar')
        Board.setBusServoPulse(1,200,delay_rutina_agarrar)
        sleep(delay_rutina_agarrar/1000)
        tomar = True
    return 'Tomar/dejar'

@app.route('/incline')
def incline():
    global inclinar
    if (inclinar == True):
        print('Inclinar hacia adelante')
        Board.setBusServoPulse(4,350,delay_rutina_inclinar)
        sleep(delay_rutina_inclinar/1000)

```

```

        inclinar = False
    else:
        print('Inclinar hacia atras')
        Board.setBusServoPulse(4,650,delay_rutina_inclinar)
        sleep(delay_rutina_inclinar/1000)
        inclinar = True
    return 'inclinarse'

@app.route('/resetServos')
def resetServos():
    Board.setBusServoPulse(1,500,1000)
    sleep(1)
    Board.setBusServoPulse(2,500,1000)
    sleep(1)
    Board.setBusServoPulse(3,150,1000)
    sleep(1)
    Board.setBusServoPulse(4,300,1000)
    sleep(1)
    Board.setBusServoPulse(5,700,1000)
    sleep(1)
    Board.setBusServoPulse(6,500,1000)
    sleep(1)
    return 'Restablecer'

@app.route('/get_sensors')
def get_battery():
    sensors[0]["value"] = Board.getBattery()
    return jsonify(sensors)

app.run(host='0.0.0.0', port=15002)

```

Código de la aplicación.

```
// llamadas directas a la API del servidor

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Net.Http;
//NuGet
//using SharpDX.DirectInput;
using SharpDX.XInput;
using AForge.Video;
using Newtonsoft.Json;

namespace AppControlRobot
{
    public partial class MainApp : Form
    {
        private bool buzzer_flag_active = false;

        private string url_motors = "http://192.168.149.1:15001/";
        private string url_servos = "http://192.168.149.1:15002/";
        private string url_sensors = "http://192.168.149.1:15002/get_sensors";

        private string url_camera = "http://192.168.149.1:8081/video_feed";

        private string url_camera_detection = "http://192.168.149.1:8081/";

        private HttpClient httpClient = new HttpClient();
```

```
private HttpClient get_sensors = new HttpClient();
```

```
private Controller controller;
```

```
private Timer buttonsTimer;
```

```
private Timer controllerTimer;
```

```
private Timer sensorsTimer;
```

```
private int buttons_motor = 10;
```

```
private int index_camera_option = 0;
```

```
private int min_value_battery = 6264;
```

```
private int max_value_battery = 8330;
```

```
MJPEGStream streamCamera;
```

```
public class Sensors_Int
```

```
{  
    public int Id { get; set; }  
    public string Type_Sensor { get; set; }  
    public string Type_Value { get; set; }  
    public int Value { get; set; }  
}
```

```
/*
```

```
public class Sensors_Float
```

```
{  
    public int Id { get; set; }  
    public string Type_Sensor { get; set; }  
    public string Type_Value { get; set; }  
    public float Value { get; set; }  
}
```

```
public class Sensors_String
```

```
{  
    public int Id { get; set; }  
}
```

```

    public string Type_Sensor { get; set; }
    public string Type_Value { get; set; }
    public string Value { get; set; }
}
*/

private string[] instructions =
{
    "up",
    "left",
    "right",
    "down",
    "diagonalUL",
    "diagonalUR",
    "diagonalBL",
    "diagonalBR",
    "rotationL",
    "rotationR",
    //Poner las instructions de los motores arriba y los de servo abajo
    //Recuerda actualizar el variable de buttons_motor
    "up",
    "left",
    "right",
    "down",
    "resetServos",
    "buzzer_on",
    "pick_leave",
    "incline"
};

private string[] camera_instructions =
{
    "cameraRaw",
    "detectRed",
    "detectGreen",

```

```

        "detectBlue"

    };

    public MainApp()
    {
        InitializeComponent();

        buttonsTimer = new Timer();
        buttonsTimer.Interval = 100; // Intervalo en milisegundos
        buttonsTimer.Tick += ButtonsTimer_Tick;

        controllerTimer = new Timer();
        controllerTimer.Interval = 150;
        controllerTimer.Tick += ControllerTimer_Tick;

        sensorsTimer = new Timer();
        sensorsTimer.Interval = 1000;
        sensorsTimer.Tick += SensorsTimer_Tick;
        sensorsTimer.Start();

        streamCamera = new MJPEGStream(url_camera);
        streamCamera.NewFrame += GetNewFrame;
        streamCamera.Start();
    }

    private void Form1_Load(object sender, EventArgs e)
    {
        controller = new Controller(UserIndex.One);

        if (controller.IsConnected)
        {
            controllerTimer.Start();
        }
    }

```

```

else
{
    buttonsTimer.Stop();
    controllerTimer.Stop();
    MessageBox.Show("No hay control conectado =(");
    Close();
    return;
}
}

private void Form1_FormClosing(object sender, FormClosingEventArgs e)
{
    controllerTimer.Stop();
    sensorsTimer.Stop();
}

private async void ButtonsTimer_Tick(object sender, EventArgs e)
{
    var button = buttonsTimer.Tag as Button; // Obtener el botón
    asociado al temporizador
    if (button != null)
    {
        int buttonIndex = int.Parse(button.Tag.ToString());
        if (buttonIndex >= 0 && buttonIndex < instructions.Length)
        {
            string url = (buttonIndex < buttons_motor) ?
                url_motors + instructions[buttonIndex] :
                url_servos + instructions[buttonIndex];

            try
            {
                await httpClient.GetAsync(url);
            }
        }
    }
}

```



```

    }
    catch (HttpRequestException ex)
    {
        controllerTimer.Stop();
        buttonsTimer.Stop();
        controllerTimer.Stop();
        buttonsTimer.Stop();

        MessageBox.Show($"Error de conexión al servidor:
{ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

        Close();
        return;
    }
}
}
}

```

```

private async void ControllerTimer_Tick(object sender, EventArgs e)
{
    var state = controller.GetState();
    var buttons = state.Gamepad.Buttons;
    var LX = state.Gamepad.LeftThumbX;
    var LY = state.Gamepad.LeftThumbY;
    var RX = state.Gamepad.RightThumbX;
    var RY = state.Gamepad.RightThumbY;
    var leftTrigger = state.Gamepad.LeftTrigger;
    var rightTrigger = state.Gamepad.RightTrigger;

    if (buttons != GamepadButtonFlags.None)
    {
        switch (buttons)
        {
            case GamepadButtonFlags.A:
                await httpClient.GetAsync(url_servos + "pick_leave");
                break;
            case GamepadButtonFlags.B:

```

```

        break;
    case GamepadButtonFlags.X:
        break;
    case GamepadButtonFlags.Y:
        break;
    case GamepadButtonFlags.DPadUp:
        break;
    case GamepadButtonFlags.DPadDown:
        break;
    case GamepadButtonFlags.DPadLeft:
        break;
    case GamepadButtonFlags.DPadRight:
        break;
    case GamepadButtonFlags.RightShoulder:
        index_camera_option += 1;
        if (index_camera_option > 3) index_camera_option = 0;
        update_color_detect();
        await httpClient.GetAsync(url_camera_detection +
camera_instructions[index_camera_option]);
        break;
    case GamepadButtonFlags.LeftShoulder:
        index_camera_option -= 1;
        if (index_camera_option < 0) index_camera_option = 3;
        update_color_detect();
        await httpClient.GetAsync(url_camera_detection +
camera_instructions[index_camera_option]);
        break;
    case GamepadButtonFlags.Start:
        await httpClient.GetAsync(url_servos + "resetServos");
        break;
    case GamepadButtonFlags.RightThumb:
        await httpClient.GetAsync(url_servos + "buzzer_on");
        buzzer_flag_active = true;
        break;
    default:
        break;

```

```

        }
    }
    else
    {
        if (buzzer_flag_active)
        {
            await httpClient.GetAsync(url_servos + "buzzer_off");
            buzzer_flag_active = false;
        }
    }
}

    if (leftTrigger == 255) await httpClient.GetAsync(url_motors +
"rotationL");
    else if (rightTrigger == 255) await httpClient.GetAsync(url_motors +
"rotationR");
    else if (LY >= 32000) await httpClient.GetAsync(url_motors + "up");
    else if (LY <= -32000) await httpClient.GetAsync(url_motors +
"down");
    else if (LX >= 32000) await httpClient.GetAsync(url_motors +
"right");
    else if (LX <= -32000) await httpClient.GetAsync(url_motors +
"left");
    else if (LX > 1000 && LY > 1000) await
httpClient.GetAsync(url_motors + "diagonalUR");
    else if (LX > 1000 && LY < -1000) await
httpClient.GetAsync(url_motors + "diagonalBR");
    else if (LX < -1000 && LY < -1000) await
httpClient.GetAsync(url_motors + "diagonalBL");
    else if (LX < -1000 && LY > 1000) await
httpClient.GetAsync(url_motors + "diagonalUL");
    else await httpClient.GetAsync(url_motors + "stopMotors");

    if (RY >= 32767) await httpClient.GetAsync(url_servos + "up");
    else if (RY <= -32767) await httpClient.GetAsync(url_servos +
"down");
    else if (RX >= 32767) await httpClient.GetAsync(url_servos +
"right");

```

```

        else if (RX <= -32767) await httpClient.GetAsync(url_servos +
"left");
    }

    private async void SensorsTimer_Tick(object sender, EventArgs e)
    {

        try
        {

            HttpResponseMessage response = await
get_sensors.GetAsync(url_sensors);

            response.EnsureSuccessStatusCode(); // Lanza una excepción si
hay error

            string json = await response.Content.ReadAsStringAsync();

            var battery_values =
JsonConvert.DeserializeObject<Sensors_Int[]>(json);

            foreach (var battery in battery_values)
            {

                //MessageBox.Show($"ID: {battery.Id}, Type_sensor:
{battery.Type_Sensor}, Type_Value: {battery.Type_Value} , Value:
{battery.Value}");

                float battery_percentage = 100 * (battery.Value -
min_value_battery) / (max_value_battery - min_value_battery);

                label_battery.Text = "Batería: " +
Convert.ToString(battery_percentage) + "%";

            }

        }

        catch (HttpRequestException error)
        {

            controllerTimer.Stop();

            buttonsTimer.Stop();

            controllerTimer.Stop();

            MessageBox.Show($"Error al realizar la solicitud HTTP:
{error.Message}");

            Close();

            return;
        }
    }
}

```

```

    }
}

private void Btn_MouseDown(object sender, MouseEventArgs e)
{
    var button = sender as Button;
    if (button != null)
    {
        buttonsTimer.Tag = button;
        buttonsTimer.Start();
    }
}

private void Btn_MouseUp(object sender, MouseEventArgs e)
{
    var button = sender as Button;
    if (button != null)
    {
        int buttonIndex = int.Parse(button.Tag.ToString());
        if (buttonIndex >= 0 && buttonIndex < buttons_motor)
        {
            try
            {
                httpClient.GetAsync(url_motors + "stopMotors").Wait();
            }
            catch (HttpRequestException ex)
            {
                controllerTimer.Stop();
                buttonsTimer.Stop();
                controllerTimer.Stop();

                MessageBox.Show($"Error de conexión al detener el
movimiento del robot: {ex.Message}", "Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);

                Close();
                return;
            }
        }
    }
}

```

```

        } else
        {
            try
            {
                httpClient.GetAsync(url_servos + "buzzer_off").Wait();
            }
            catch (HttpRequestException ex)
            {
                controllerTimer.Stop();
                buttonsTimer.Stop();
                controllerTimer.Stop();
                buzzer: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
                Close();
                return;
            }
        }
    }
    buttonsTimer.Stop();
}

private void GetNewFrame(object sender, NewFrameEventArgs eventArgs)
{
    Bitmap bmpvideo = (Bitmap)eventArgs.Frame.Clone();
    camara.Image = bmpvideo;
}

private void btn_raw_Click(object sender, EventArgs e)
{
    httpClient.GetAsync(url_camera_detection +
camera_instructions[0]).Wait();
    index_camera_option = 0;
    update_color_detect();
}

private void btn_red_Click(object sender, EventArgs e)

```

```

        {
            httpClient.GetAsync(url_camera_detection +
camera_instructions[1]).Wait();
            index_camera_option = 1;
            update_color_detect();
        }

        private void btn_green_Click(object sender, EventArgs e)
        {
            httpClient.GetAsync(url_camera_detection +
camera_instructions[2]).Wait();
            index_camera_option = 2;
            update_color_detect();
        }

        private void btn_blue_Click(object sender, EventArgs e)
        {
            httpClient.GetAsync(url_camera_detection +
camera_instructions[3]).Wait();
            index_camera_option = 3;
            update_color_detect();
        }

        private void update_color_detect()
        {
            switch (index_camera_option)
            {
                case 0:
                    pictureBox_color_detect.BackColor = Color.Transparent;
                    break;
                case 1:
                    pictureBox_color_detect.BackColor =
Color.FromArgb(255,50,50);
                    break;
                case 2:
                    pictureBox_color_detect.BackColor =
Color.FromArgb(90,220,90);

```

```
        break;
    case 3:
        pictureBox_color_detect.BackColor =
Color.FromArgb(0,120,180);
        break;
    }
}

protected override void Dispose(bool disposing)
{
    if (disposing)
    {
        MessageBox.Show("Recursos liberados");
        httpClient.Dispose();
        get_sensors.Dispose();
    }
    base.Dispose(disposing);
}
}
```