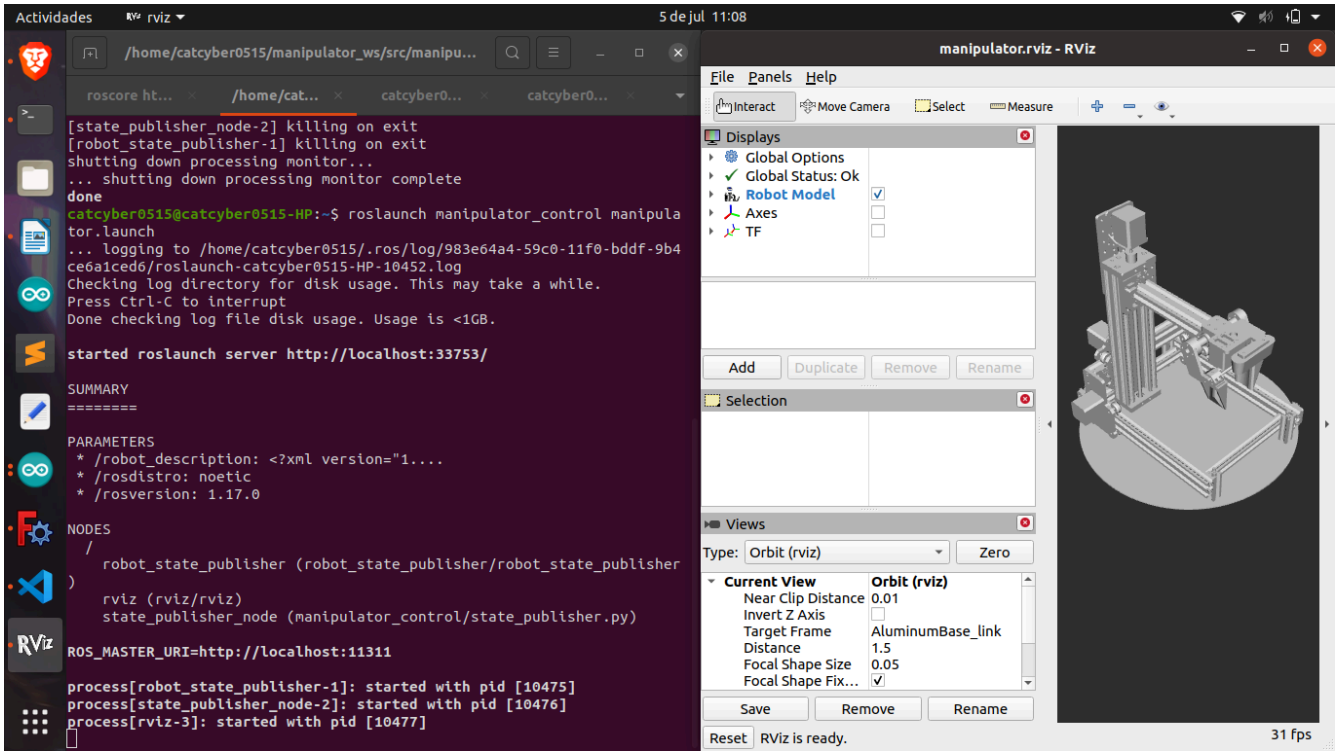
Documentación Manipulador.

1) Abrir Terminal de Linux y ejectuar el comando ***roscore.***

2) En otra ventana correr ejecutar ***roslaunch manipulator_control manipulator.launch***

Se cargara la simulación en rviz,



3) En otra ventana ejecutar ***rosrun manipulator_control serial_interface.py***
Nota: Verificar la dirección del puerto serial del ESP32.

self.ser = serial.Serial('/dev/ttyUSB0', 115200, timeout=1)

4) Comandos para mover interactuar con el manipulador:

- Posición en milímetros x,y,z.
  rostopic pub /manipulator_move geometry_msgs/Point "{x: 100.0, y: 50.0, z: 20.0}"
- Para abrir el gripper.
  rostopic pub /manipulator_gripper std_msgs/Bool "data: true"
- ***Para cerrar el gripper***
  rostopic pub /manipulator_gripper std_msgs/Bool "data: false"
- ***Para regresar al 0,0,0 origen.***
  rostopic pub /manipulator_home std_msgs/Bool "data: true"

Configuración del ESP32.

Para usar el manipulador primero es necesario crear un objeto de la clase manipulator ejemplo:

MANIPULATOR_SHIELD festo_Manipulator;

Después en el setup configurar su funcionamiento.

```
void setup() {
  …….
  festo_Manipulator.init();
  festo_Manipulator.set_XYZ_STEPmm(5, 5, 25);            // 20 20 100 para 1/4 paso.
  festo_Manipulator.set_XYZ_maxRPM(200, 200, 450);  // 200 rpm 200 rpm 450 rpm
  festo_Manipulator.set_minSpeedPercentage(20);           // 20%
  ……
}
```

init()

El método init(), inicializa la configuración de pines, y inicializa los finales de carrera, “Nota los finales de carrera funcionan por medio de hilos, en el core 1 del ESP32”

set_XYZ_STEPmm(double Xstp_mm, double Ystp_mm, double Zstp_mm)

El método inicializa los pasos por milímetro para X, Y, Z, para saber este dato en caso de cambiar la correa o el husillo, se deben aplicar las siguientes fórmulas:

1) Para Z que tiene un Husillo:

   Datos:

- Motor: NEMA 17 → **200 pasos por revolución**
- Microstepping → **paso completo “si se usa un ¼ Microstepping = 4 ”**
- Husillo: T8 → **8 mm de avance por revolución**

Aplicando la fórmula:

$$Pasos\ por\ mm\ = \frac{Pasos\ por\ revolución^{*}\ Microstepping}{Avance\ por\ revolución\ (mm)}\ = \frac{200^{*}1}{8} = 25\ pasos/mm$$

2) Para X, Y que usan una polea dentada:
   Datos:

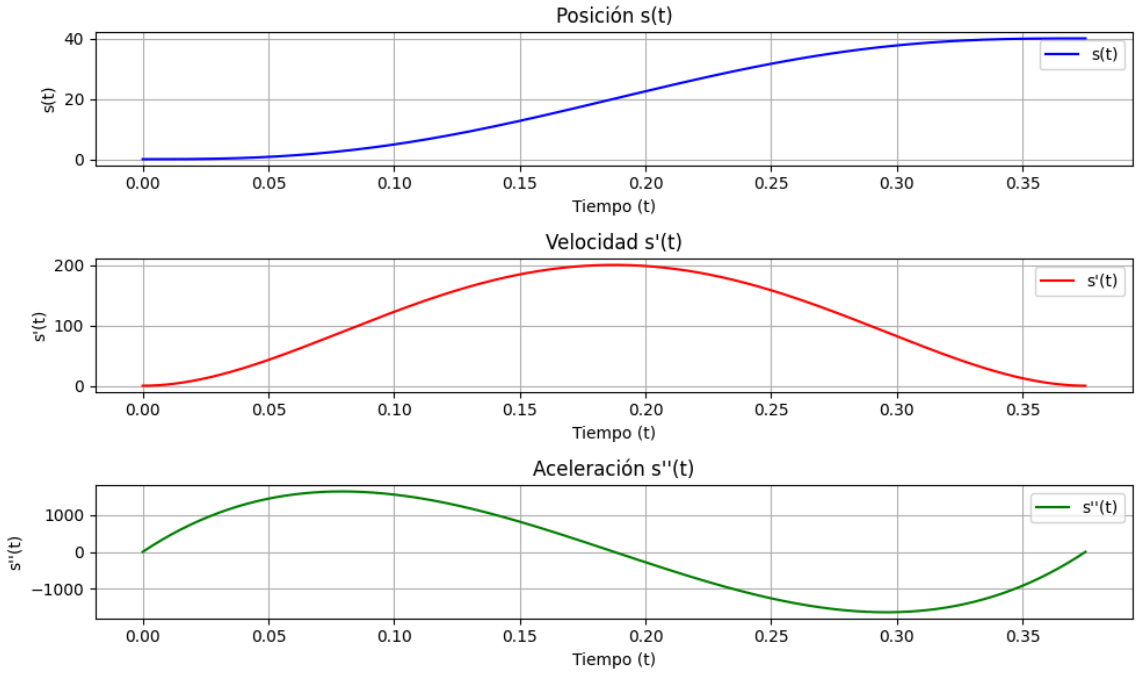
- Correa: GT2 (2 mm por diente)
- Polea: 20 dientes
- Motor: NEMA 17 (200 pasos por revolución)
- Microstepping: 1 paso completo.

Fórmula:

$$Pasos\ por\ mm\ =\ \frac{Pasos\ por\ revolución\ *\ Microstepping}{Paso\ de\ correa^{*}Dientes\ de\ la\ polea} = \frac{200^{*}1}{20^{*}2} = 5\ pasos/mm$$

set_XYZ_maxRPM (double _XmaxRPM, double _YmaxRPM, double _ZmaxRPM).

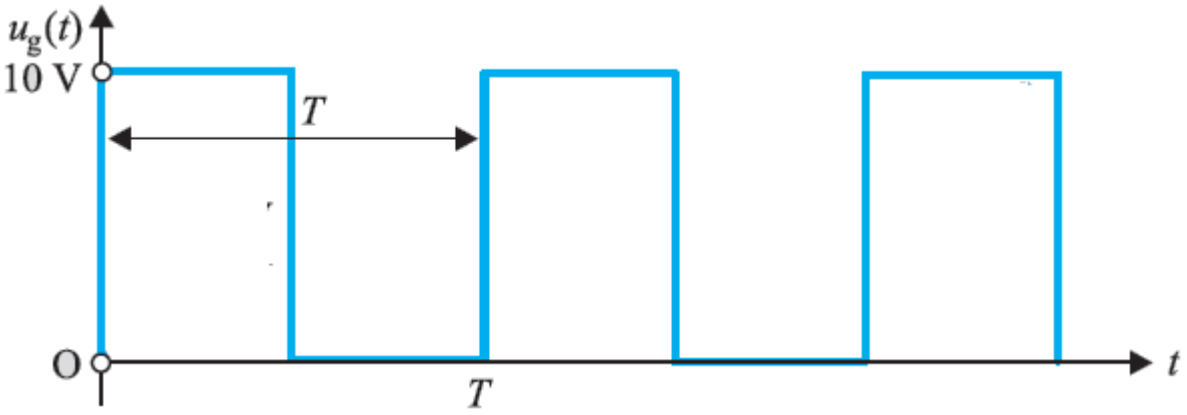
Este método inicializa la velocidad máxima que puede alcanzar cada motor a pasos, de acuerdo al un perfil de velocidades polinomial de quinto orden.

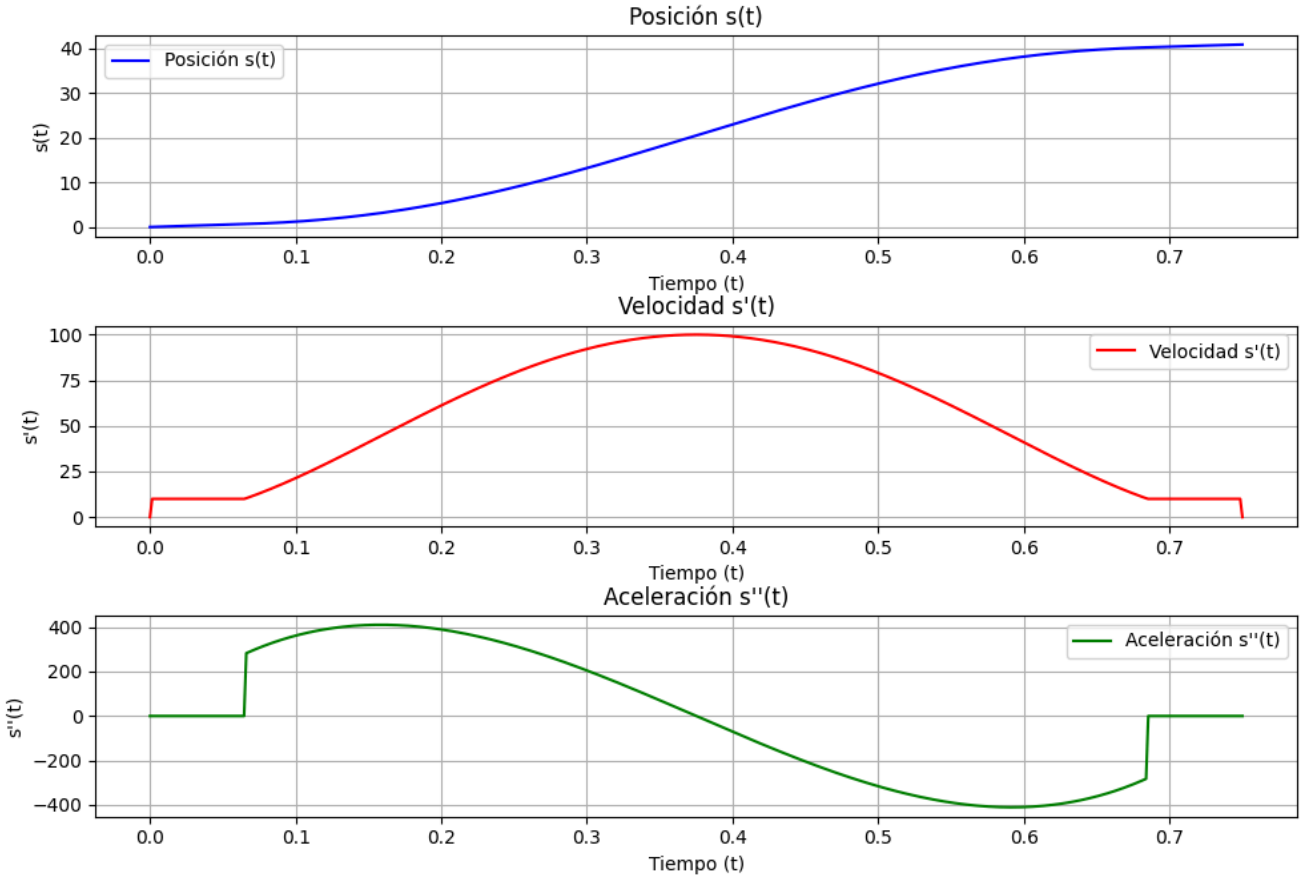


Para este ejemplo la velocidad máxima será 200 rpm, y será 0 al llegar a los 40 mm.

set_minSpeedPercentage (double percent).

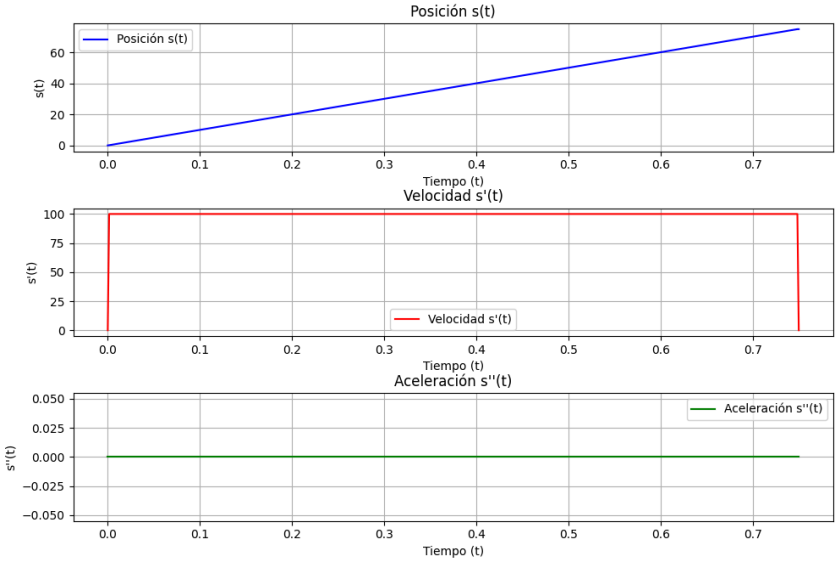
Como bien se sabe los motores a pasos funcionan a base de señales cuadradas, donde la frecuencia de dicha señal es el parámetro que marca la velocidad a la que irá nuestro motor.
Sin embargo para el caso de la velocidad 0 o cercanas al 0 la frecuencia de dicha señal sería f=0 y T→∞, lo que causa problemas de interpretación para el ESP32, movimientos raros, pérdidas de pasos y torque; es por ello que no se parte de una velocidad 0 si no de un valor en % de la velocidad máxima.





Para este ejemplo para un velocidad de 100 rpm, se pare de 10 rpm,

Nota: Si se desea trabajar de manera lineal poner este valor al 100 %



Nota: Este porcentaje aplica para todos los motores. Es decir de momento no es posible que uno esté al 100 % y otro al 10 %, aplica para todos.

move_X (double mm)
Este método mueve al eje X cierta distancia en milímetros -mm para retroceder +mm para avanzar, detiene el avance al detectar el final de carrera.

move_Y (double mm)
Este método mueve al eje Y cierta distancia en milímetros -mm para retroceder +mm para avanzar, detiene el avance al detectar el final de carrera.

move_Z (double mm)
Este método mueve al eje Z cierta distancia en milímetros -mm para retroceder +mm para avanzar, detiene el avance al detectar el final de carrera.

move_XYZ (double Xmm, double Ymm, double Zmm);
Este método mueve al eje XYZ simultáneamente cierta distancia en milímetros -mm para retroceder, +mm para avanzar, detiene el avance en el eje al detectar el final de carrera.

move_Home ( );
Este método mueve al eje XYZ, al punto 0,0,0.

get_FRX ( )
Retorna un bool con el estado del final de carrera X.

get_FRY ( )

Retorna un bool con el estado del final de carrera Y.

get_FRZ ( )
Retorna un bool con el estado del final de carrera Z.

move_Claw (bool dir , uint16_t vel, long _t)

Este método mueve la garra del manipulador, en un sentido, a una velocidad en rpm, y un cierto tiempo.

Nota: Pasar el límite de la garra puede dañar el servo y el mecanismo.

Ejemplo de uso.

```
#include "Manipulator_Shield.h"

MANIPULATOR_SHIELD festo_Manipulator;

void setup() {
  Serial.begin(115200);
  Serial.println("Iniciando Manipulador...");
  festo_Manipulator.init();
  festo_Manipulator.set_XYZ_STEPmm(5, 5, 25);            // 20 20 100 para 1
  festo_Manipulator.set_XYZ_maxRPM(200, 200, 450);  // 200 rpm 200 rpm 450 rpm
  festo_Manipulator.set_minSpeedPercentage(20);           // 15%
  Serial.println("Manipulador listo.");
}

void loop() {
  Serial.println("Moviendo eje X +100 mm...");
  festo_Manipulator.move_X(100.0);
  delay(100);

  Serial.println("Moviendo eje Y +50 mm...");
  festo_Manipulator.move_Y(50.0);
  delay(100);

  Serial.println("Moviendo eje Z +30 mm...");
  festo_Manipulator.move_Z(+30.0);
```

```
  delay(100);

  Serial.println("Moviendo eje X -100 mm.  Y -50 mm. Z -30 mm.");
  festo_Manipulator.move_XYZ( -50.0, -25, -15 );
  delay(100);

  Serial.println("Moviendo garra...");
  festo_Manipulator.move_Claw(true, 200, 1000); // Direccion: true, Vel: 200, Tiempo: 1000 ms
  delay(100);

  Serial.println("Regresando a Home...");
  festo_Manipulator.move_Home();
  delay(2000);

  Serial.println("Movimiento completo. Repetir en 0.5 segundos.");
  delay(500);
}
```