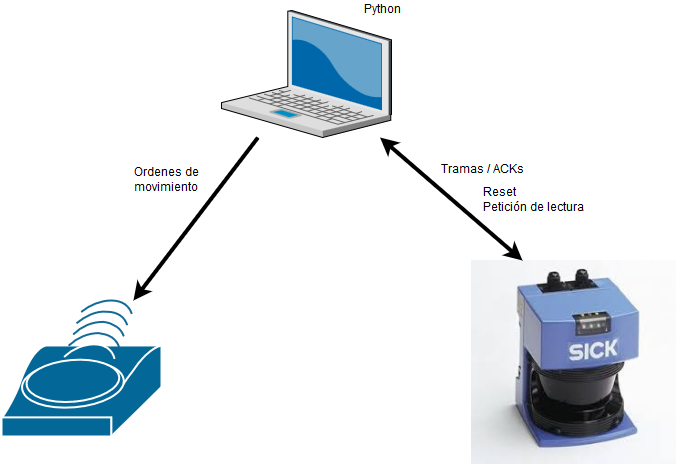
# Arquitectura del sistema





El programa está en la versión 3.6 de Python ya que es la versión más actualizada con las librerías funcionando correctamente, estas son:

* Time: para hacer las pausas (sleeps)
* Numpy: creacion de array y estructuras
* Math: operaciones de seno y coseno
* Matplotlib: mostrar una ventana con las lecturas del sensor con la sintaxis de MatLab
* Serial: comunicación por los puertos USB

# Versiones de los programas

Para manejar Lidar se han realizado tres versiones de programa. El primer intento fue en MatLab, funciona correctamente, pero al no ser la posible la integración del robot a este entorno quedó como un ensayo guardado en el archivo *“SICK\_primer\_contacto.m”.* El segundo intento fue en C++, también funciona correctamente, el proyecto se encuentra en la carpeta *“LecturaSick\_C++”* para Visual Studio,pero tampoco se continuó por aquí por la imposibilidad de hacer un programa en el mismo lenguaje para el robot que funcionase. Por último, se realizó su programación en Python, este proyecto está guardado en la carpeta *“SICK\_Python”* para el entorno de Anaconda.

Contando con el programa de otro estudiante que fue capaz de mover el robot con un joystick conectado a un arduino (*“RoMax\_Joystick\_Python”*), se realizó un proyecto en Visual Studio que une ambos programas de Python, como resultado salió el proyecto *“LIDAR\_RoMax\_Python”,* siendo éste el programa sobre el que se trabaja.

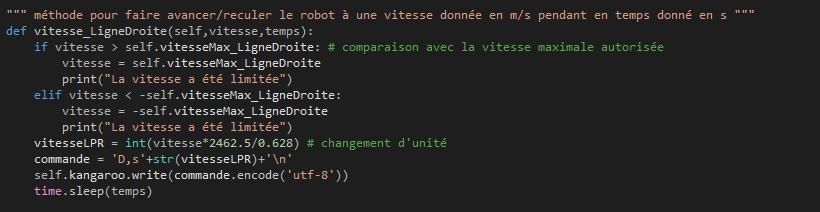
# Proyecto LIDAR\_RoMax\_Python

**Main.py**

Para funcionar usa las clases: classRobot y sick. Toma 6 imágenes del sensor y hace un filtro de mediana con las cinco primeras, dejando la sexta para sobreescribir con el resultado del filtro. Ese mapa se compara con una séptima imagen en la que se supone que habrá un objeto que dependiendo donde se encuentre se le enviará una orden al robot de movimiento en una dirección u otra.

**ClassRobot.py**

Contiene las funciones de inicialización y movimiento del robot, ya sea con una consigna de velocidad o de posición. Por ejemplo:

Esta función es llamada desde el main con los parámetros de velocidad (vitesse), que admite valores con decimales positivos y negativos y un tiempo (temps) que admite sólo valores positivos. El valor de velocidad introducido es pasado a través de un factor de conversión para que el software de la placa del robot lo entienda, este factor se sabe gracias al software *DEScribe,* que se conecta con la placa para configurar parámetros (se recomienda no modificar nada). Como resultado se obtiene una cadena del tipo ”D,s(valor)”, significa que va a avanzar o retroceder a cierta velocidad (para otras funciones se puede consultar la documentación de la placa Sabertooth).

**Sick.py**

Contiene las funciones de inicialización y creación de tramas para el sensor. La creación de las tramas se realiza por bloques según lo que se quiera hacer, una vez acabada su creación se envía por el puerto USB correspondiente, siendo comprobada la validez de las tramas mediante el cálculo del *checksum* mediante llamadas al programa *CRC16\_SICK*. La otra función más relevante que contiene es *calc\_distances*, a la que se introduce la trama de coordenadas devuelta por el sensor y se interpreta, devolviendo un vector formado por coordenadas cartesianas en el plan x – y junto con su módulo.

# Mejoras a futuro

Dado el tiempo de retardo que se encuentra entre las tomas por parte del SICK y el tratamiento de los datos en python, se ha llegado a la concusión que el proyecto debería de dividirse en dos ramas.

La primera, que buscase la optimización por parte del código proveniente del SICK y su posterior tratamiento, ya que se ha observado que el cálculo del ***checksum*** tarda demasiado y es necesario la modificación del mismo si se quiere que la realización del seguimiento se realice de forma eficiente. El equipo que se encargase de esta labor no solo se encargaría de esto, ya que existen problemas añadidos por parte de las librerías de python. Tal es el caso de la librería ***matplotlib*** la cual permite la visualización del mapa creado, pero es necesario el cierre de este mapa cada vez que se genera, ralentizando el proceso de esta manera y no permitiendo la detección eficiente del objetivo. Las pruebas finales de movimiento se han realizado sin la muestra gráfica del mapa, por lo que deberán de obtener una forma de visualización.

Una forma sería el añadido de las librerías de ***opencv*** que permite ***VisualStudio,*** ya que se han encontrado ejemplos en internet de esta realización, pero no se ha podido finalmente llevar a cabo la representación gráfica con esta librería.

La segunda, que buscase la modificación y la optimización del algoritmo de seguimiento. Ahora mismo, el robot se moverá a la derecha si se detectan movimientos del objetivo entre los 0º y los 45º, hacia delante si se detectan entre 45º y 135º y hacia la izquierda si el ángulo es mayor de 135º.

Este equipo deberá de buscar la optimización de este algoritmo para llevar a cabo el movimiento del robot.