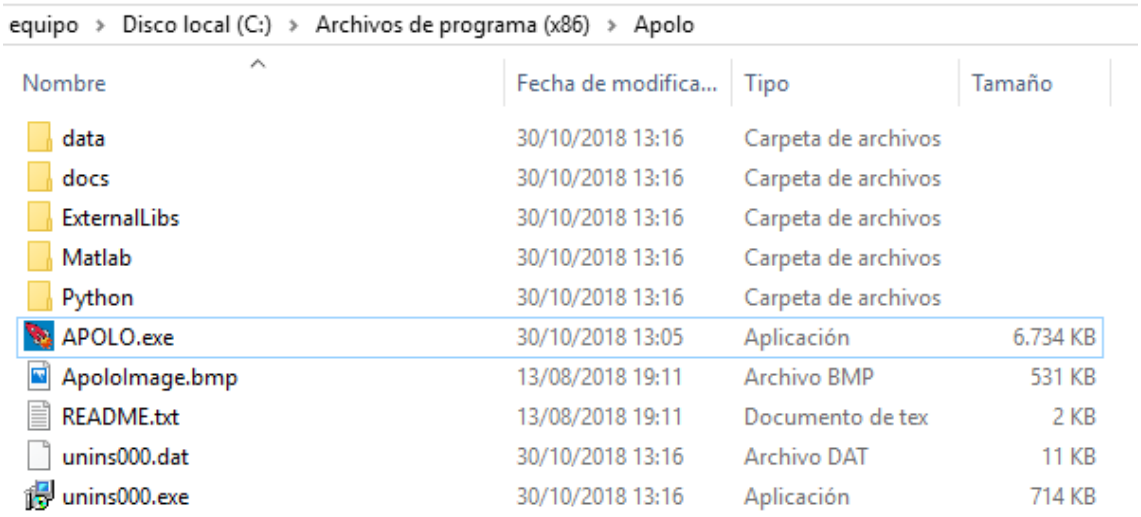
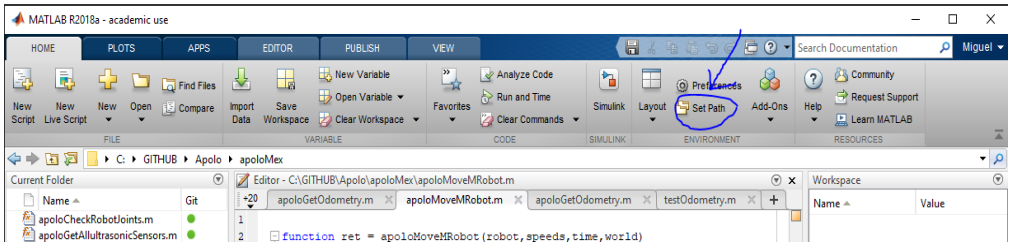
EJEMPLO DE USO DE APOLO Y MATLAB

- Instalar Apolo. Es una versión de 64 bits, compilada sobre Windows 10 y con Matlab de 64 bits también.
- Al instalarlo, se crea la siguiente estructura:

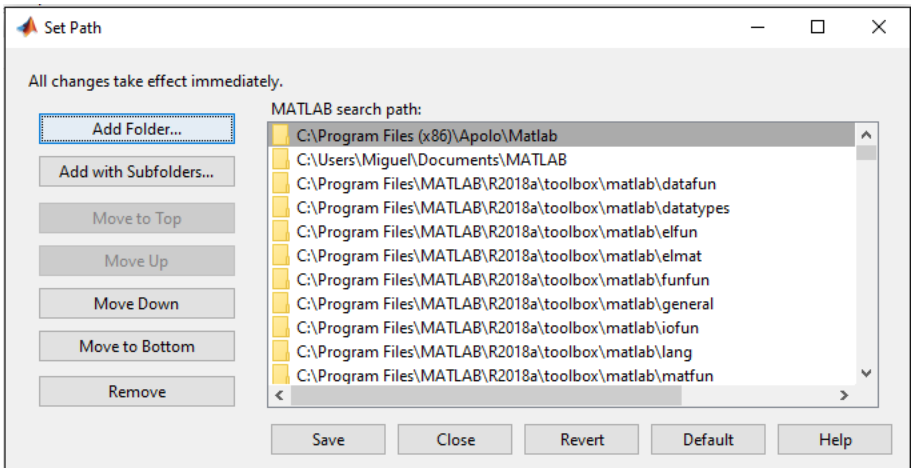


El programa no toca el registro salvo para agregar la aplicación al menú por lo que se puede desinstalar con un borrado (y quitar el acceso directo del menú de aplicaciones). De todas formas, se incluye un desinstalador que lo deja todo limpito.

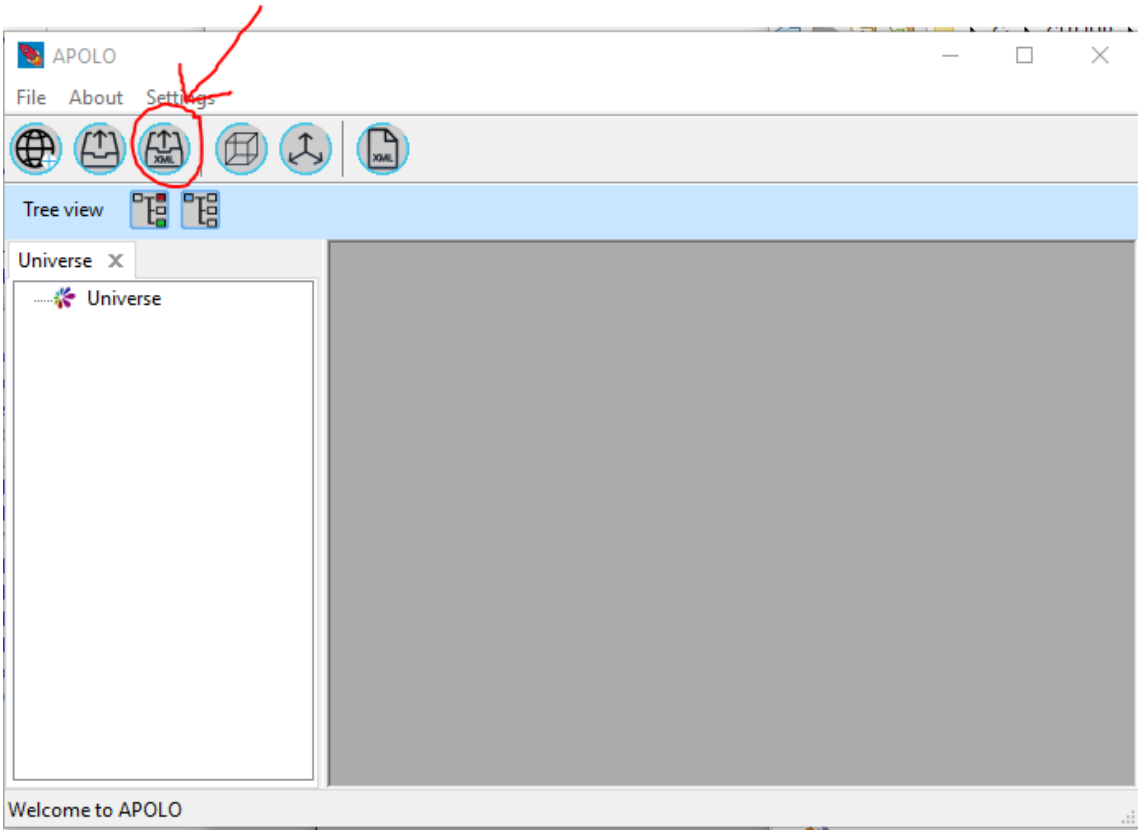
- Para poder ejecutar los comandos de Matlab, es conveniente agregar al path de Matlab el directorio Matlab de Apolo. La otra opción es copiarse esa carpeta en donde tengamos nuestro script (no recomendado).



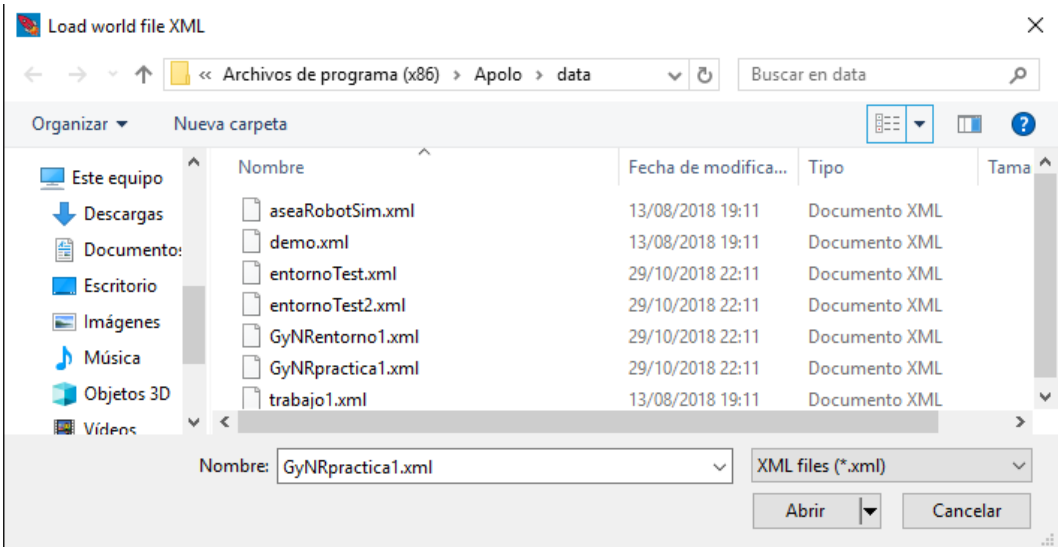
- Agregamos el directorio Matlab de Apolo



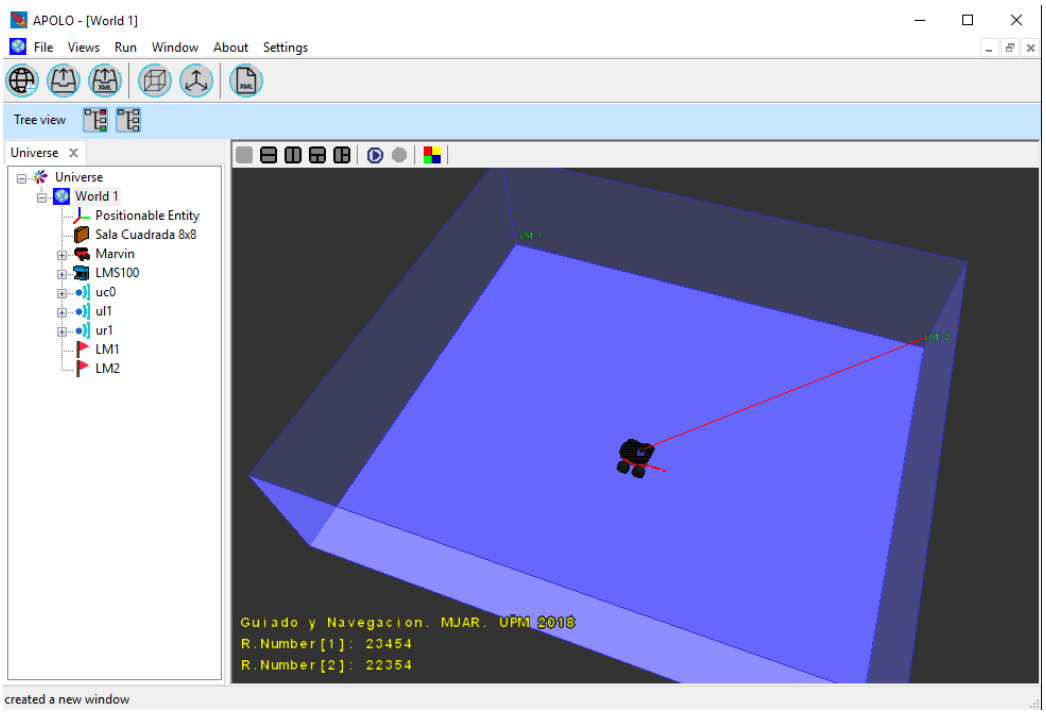
- Abrimos Apolo directamente desde el menú de Windows. Y abrimos un entorno definido mediante un fichero XML. Esto es así porque así podemos editar en cualquier editor de texto la información del entorno. Al haberse incluido muchas modificaciones, la mayoría de estas no es posible editarlas por medio de Apolo, pero están disponibles todas en XML directamente.



- Seleccionamos el archivo GyNRpractica1.xml incluido en la instalación en el directorio Apolo/data:



- Es un entorno en el que se ha incluido un Sick LMS100 montado sobre un Pioner3AT en una sala cuadrada con dos balizas. Además se han agregado 3 sensores de ultrasonidos al robot, y la información de los números de matrícula de los alumnos del trabajo. Aunque aparecen reflejados.



- Si abrimos el fichero XML en un editor de texto (Notepad ++, Word, o un navegador) podemos ver su estructura. Consideramos que es bastante explícito. Las medidas se dan siempre en metros y los ángulos en radianes. Los colores en valores de componentes rgb entre 0 y 1, y las orientaciones son ángulos de cardan correspondientes al producto matricial: $Rot(x,\alpha)Rot(y,\beta)Rot(z,\gamma)$. Veamos algunos elementos relevantes para el trabajo:
    - **WorldInfo**: la información sobreimpresa que además afecta a la varianza de los sensores del robot.

```
<!-- Los alumnos deben poner el número entero de su número de matrícula a continuación -->
<WorldInfo name="Guiado y Navegacion. MUAR. UPM 2018">
    <RegisterNumber number="23454"/>
    <RegisterNumber number="22354"/>
</WorldInfo>
```

      Los alumnos incluirán tantos RegisterNumber como números de matrícula. El atributo number es el número entero correspondiente. Podría sustituirse por un DNI (sin letra) etc. Dado un conjunto de números de matrículas las varianzas de los sensores y odometría son biunívocas.

    - **Pinoeer3ATSim**: Es el robot, que daremos el nombre que queramos. Existen más robots móviles, pero el que está más validado en el simulador es este. Sus características físicas son las mismas que las del robot real.

```
<!-- definición de un robot Pioner3at , posicion y orientación inicial-->
    <Pioneer3ATSim name="Marvin">
      <orientation> {0,0,0} </orientation>
      <position>    {0,0,0} </position>
    </Pioneer3ATSim>
```

    - **LMS100Sim**: Es un Sick LMS100 con sus características de rango y resolución. Es interesante ver que se enlaza con una determinada posición al robot Marvin. Esto se realiza con el atributo linkTo=”$Marvin$”.

```
<!-- definición de un laser unido al robot , posicion y orientación relativa-->
    <LMS100Sim name="LMS100" linkTo="$Marvin$">
      <position>     {0.1,0,0.4} </position>
      <colour r="0" g="1" b="0"/>
    </LMS100Sim>
```

Existen dos sensores laser perfectamente usables. Todos ellos quedarán modificados por el WorldInfo en cuanto a la varianza de sus medidas. Sensores disponibles: LMS100Sim, LMS200Sim.

- **UltrasonicSensor**: De igual forma se pueden definir Sensores de Ultrasonidos. Por defecto adoptan un valor de cono y rango predefinido, pero todo ello es variable.

```
<!-- definición de 3 ultrasonidos frontales unidos al robot , posicion y orientación relativa-->
    <UltrasonicSensor name = "uc0" linkTo="$Marvin$">
      <orientation> {0,-0.1,0}  </orientation>
      <position>    {0.2,0,0.2} </position>
      <colour r="0" g="1" b="0"/>
    </UltrasonicSensor>
```

- **LandMark**: Es el modo en que se modelan las balizas. Las balizas deben estar referidas al mundo y se puede incluir una información identificativa (mark_id). Si se quisieran balizas sin identificación bastaría con no poner este atributo.

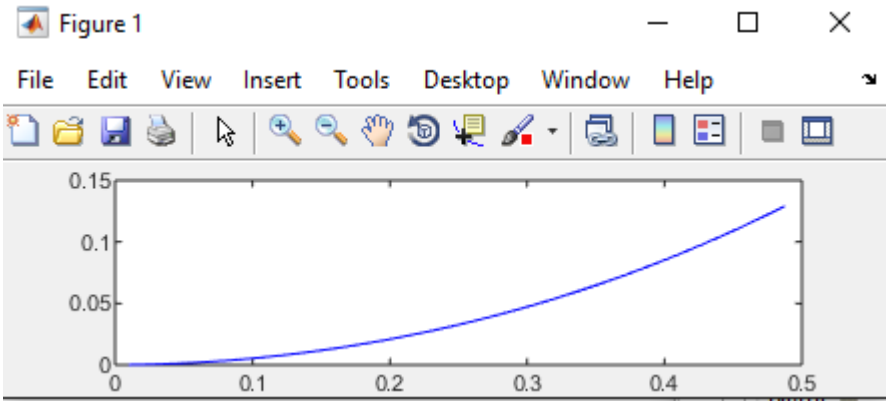
```
<!-- definición de 2 balizas-->
    <LandMark  name ="LM1" mark_id="1">
          <position> {-3.9,3.9,0.2} </position>
    </LandMark>

    <LandMark  name = "LM2" mark_id="2">
          <position> {3.9,3.9,0.2} </position>
    </LandMark>
```

- El robot en Apolo se puede mover fácilmente activando la simulación, y ejecutando del menú contextual de Marvin , el comando Move Control. En cualquier caso, desde Matlab, es controlable completamente. Importante: siempre que lancemos un comando desde Matlab, la simulación se parará para evitar problemas debido a la sección crítica de los datos de simulación.
- Ejemplo de Script de Matlab que mueve el robot y recoge los datos de la odometría:

```
pos=apoloGetOdometry('Marvin')
A=[pos]
for i = 1:100
    apoloMoveMRobot('Marvin', [0.05, 0.05], 0.1);
    apoloUpdate()
    a=apoloGetOdometry('Marvin');
    A=[A ; a];
end
plot(A(:,1), A(:,2),'b')
```

- **apoloUpdate** es simplemente para refrescar la vista. Los datos de los sensores siempre se actualizan automáticamente cuando se solicitan
- **`apoloMoveMRobot`**, mueve el robot a una velocidad dada por el vector `([0.05 m/s, 0.05 rad/s])` durante un tiempo indicado `(0.1s )`.
- Toda la información específica de los comandos, se incluye en el directorio doc de la instalación.

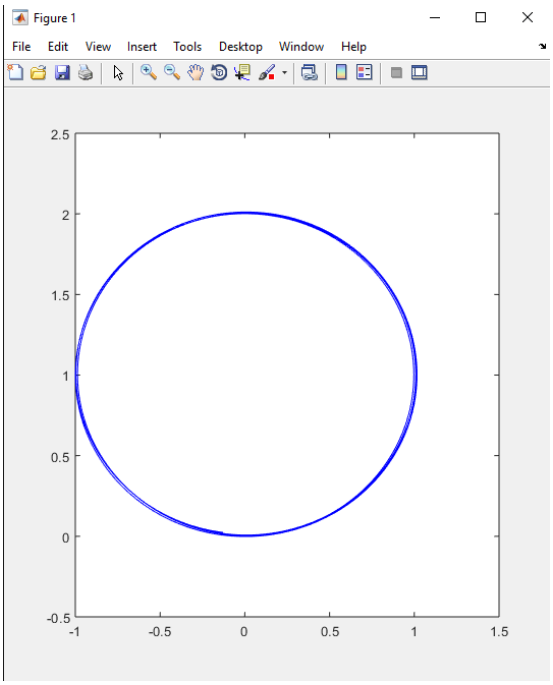
El resultado de ejecutar el script es el siguiente:



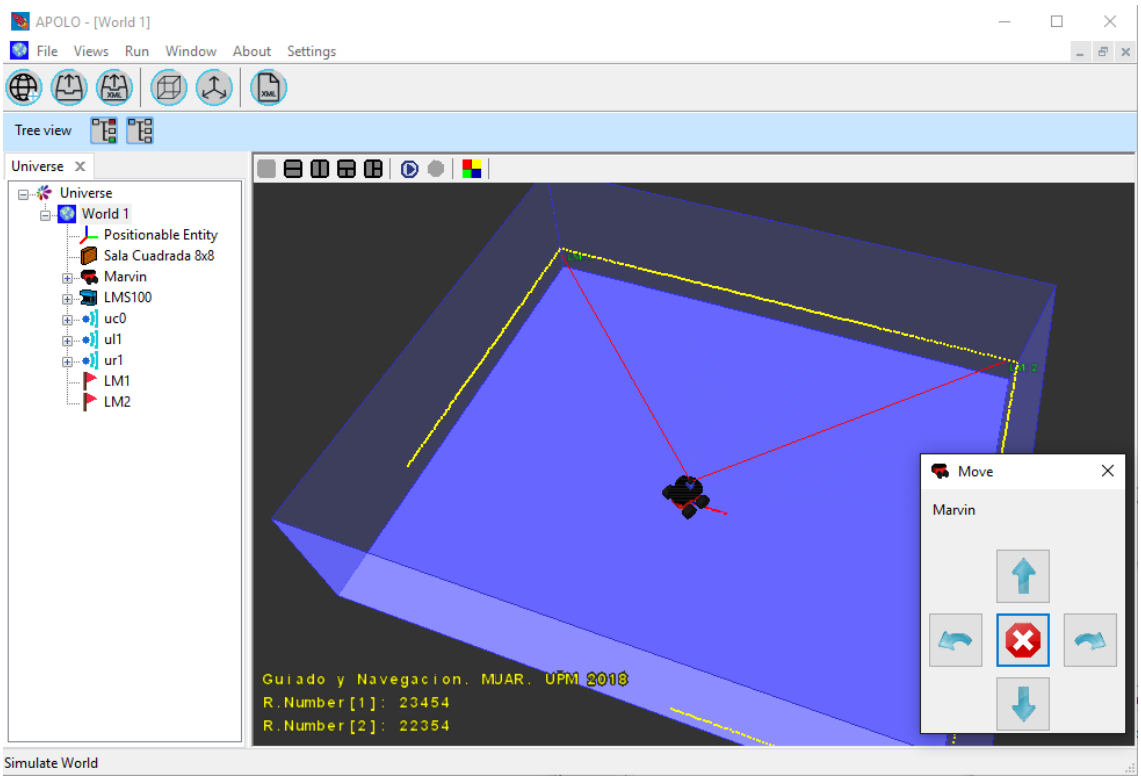
- Si quisiéramos recolocar el robot, deberíamos reposicionarlo y resetear la odometría:

```
>> apoloPlaceMRobot('Marvin',[0 0 0],0)
>> apoloResetOdometry('Marvin')
>> apoloUpdate
```

Si ejecutamos el mismo código pero ahora con 5000 iteraciones, se observa el ruido en la odometría:



La obtención de las medidas laser es similar. Para verlo, hemos movido el robot ligeramente de forma que se ven las dos balizas:



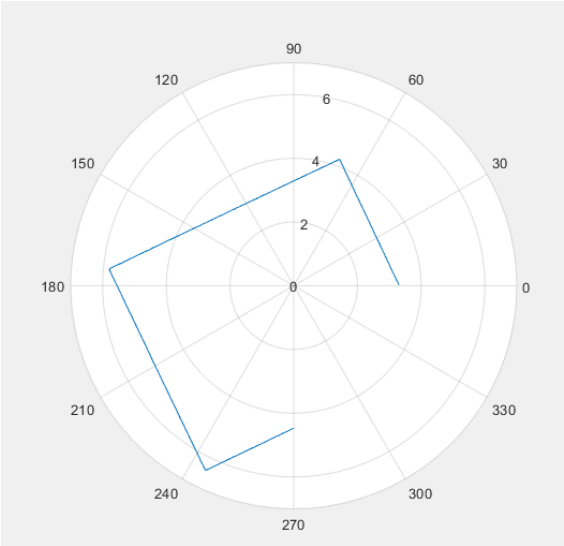
- Vamos a ejecutar un código que permita la visualización del laser. Para ello abrimos y vemos el script incluido en los ejemplos de Matlab *testLaserData*:

```
function A= testLaserData(laser,robot,n)
%TESTLaserData
      for i = 1:n
            apoloMoveMRobot(robot, [0.1, 0.2], 0.1);
            a=apoloGetLaserData(laser);
            b=size(a);
            t = 1:b(2);
            t = t*(1.5*pi/b(2));
            polarplot(t, a);
            pause(0.5)
            apoloUpdate()
      end
      A=a
end
```

Su ejecución nos permite ir viendo cada medio segundo las capturas del sensor laser.

```
>> a=testLaserData('LMS100','Marvin',100)
```

Por supuesto que se puede ir más rápido. El sistema está protegido aunque se intente "abrumarlo" con peticiones (mientras el ordenador tenga memoria claro).
El script muestra la evolución de las medidas del sensor mientras se mueve el robot:



- Finalmente la obtención de las balizas, es muy directo. Por ejemplo la ejecución de:
  ```
  >> a=apoloGetLaserLandMarks('LMS100')
  ```
  Nos da la siguiente estructura:
  ```
  a =

    struct with fields:

            id: [1 2]
         angle: [1.5753 0.0128]
      distance: [4.7722 6.1232]
  ```