ENTREGABLE 23:
INCORPORACIÓN DE LA NAVEGACIÓN AUTÓNOMA A LA SOLUCIÓN ROBÓTICA MÓVIL

(Octubre 2022)

ACTIVIDAD 3: DESARROLLO DE CAPACIDADES FORMATIVAS Y ENTORNOS DE EXPERIMENTACIÓN PARA LA INNOVACIÓN DIGITAL EMPRESARIAL

Contenido

1. OBJETIVO TAREA

En este informe se detalla los algoritmos de navegación implementados en la solución robótica móvil. Estos algoritmos permiten la negación empleando exclusivamente el LiDAR e IMU y mapeando previamente la zona usando SLAM o incorporar el GPS para seguir una determinada trayectoria. Además se detallarán algoritmos empleados para la detección de personas y objetos.

2. ALGORITMOS DE NAVEGACIÓN IMPLEMENTADOS

La solución robótica, que cuenta con un modo manual mediante un control remoto, no presentará este modo de funcionamiento de forma general sino que su aplicación está orientada al movimiento de forma autónoma, ya sea con mapas precargados, generados mediante la sensórica instalada o por GPS. Esta ultima será la preferida para navegar por un entorno real en exterior, como una parcela de cualquier cultivo. Para ello se usará el paquete *move_base*. La estructura del archivo *move_base_odom.launch* se puede ver a continuación:

```
<?xml version="1.0"?>
<!-- NEW SUMMIT XL NAVIGATION -->
<launch>

    <arg name="no_static_map" default="true"/>

    <arg name="base_global_planner" default="navfn/NavfnROS"/>




    <arg name="base_local_planner"
default="dwa_local_planner/DWAPlannerROS"/>
    <!-- <arg name="base_local_planner"
default="base_local_planner/TrajectoryPlannerROS"/> -->

    <node pkg="move_base" type="move_base" respawn="false"
name="move_base" output="screen">

        <param name="base_global_planner" value="$(arg
base_global_planner)"/>
        <param name="base_local_planner" value="$(arg
base_local_planner)"/>



<rosparam file="$(find my_summit_xl_tools)/config/planner.yaml"
command="load"/>
```

```
        <!-- observation sources located in costmap_common.yaml -->
        <rosparam file="$(find
my_summit_xl_tools)/config/costmap_common.yaml" command="load"
ns="global_costmap" />
        <rosparam file="$(find
my_summit_xl_tools)/config/costmap_common.yaml" command="load"
ns="local_costmap" />

        <!-- local costmap, needs size -->
        <rosparam file="$(find
my_summit_xl_tools)/config/costmap_local.yaml" command="load"
ns="local_costmap" />
        <param name="local_costmap/width" value="5.0"/>
        <param name="local_costmap/height" value="5.0"/>

        <!-- static global costmap, static map provides size -->
        <rosparam file="$(find
my_summit_xl_tools)/config/costmap_global_static.yaml"
command="load" ns="global_costmap" unless="$(arg no_static_map)"/>

        <!-- global costmap with laser, for odom_navigation_demo --
>
        <rosparam file="$(find
my_summit_xl_tools)/config/costmap_global_laser.yaml"
command="load" ns="global_costmap" if="$(arg no_static_map)"/>
        <param name="global_costmap/width" value="100.0" if="$(arg
no_static_map)"/>
        <param name="global_costmap/height" value="100.0" if="$(arg
no_static_map)"/>

        <remap from="/cmd_vel" to="/summit_xl_control/cmd_vel" />
    </node>

</launch>
```

El paquete *move_base* usa archivos de configuración en formato *.yaml* que definen qué planificador se va a utilizar, cómo se utilizan los mapas o los sensores incorporados. El archivo costmap_common.yaml presenta la siguiente estructura:

```
footprint: [[0.35, -0.3], [0.35, 0.3], [-0.35,0.3], [-0.35, -0.3]]
footprint_padding: 0.01

robot_base_frame: summit_xl_a_base_link
update_frequency: 4.0
publish_frequency: 3.0
transform_tolerance: 0.5

resolution: 0.05

obstacle_range: 5.5
```

```
raytrace_range: 6.0

#layer definitions
static:
    map_topic: /map
    subscribe_to_updates: true

obstacles_laser:
    observation_sources: hokuyo_laser
    hokuyo_laser: {sensor_frame: summit_xl_a_front_laser_link,
data_type: LaserScan, clearing: true, marking: true, topic:
hokuyo_base/scan, inf_is_valid: true}

inflation:
    inflation_radius: 1.0
```

El archivo *planner.yaml* contiene el planificador a usar (NavfnROS para la navegación global y TrajectoryPlannerROS o DWAPlannerROS para la local) y presenta la siguiente estructura:

```
controller_frequency: 5.0
recovery_behaviour_enabled: true

NavfnROS:
  allow_unknown: true # Specifies whether or not to allow navfn to
create plans that traverse unknown space.
  default_tolerance: 0.1 # A tolerance on the goal point for the
planner.

TrajectoryPlannerROS:
  # Robot Configuration Parameters
  acc_lim_x: 2.5
  acc_lim_theta:  3.2

  max_vel_x: 1.0
  min_vel_x: 0.0

  max_vel_theta: 1.0
  min_vel_theta: -1.0
  min_in_place_vel_theta: 0.2

  holonomic_robot: false
  escape_vel: -0.1

  # Goal Tolerance Parameters
  yaw_goal_tolerance: 0.1
  xy_goal_tolerance: 0.2
  latch_xy_goal_tolerance: false

  # Forward Simulation Parameters
  sim_time: 2.0
  sim_granularity: 0.02
```

```
  angular_sim_granularity: 0.02
  vx_samples: 6
  vtheta_samples: 20
  controller_frequency: 20.0

  # Trajectory scoring parameters
  meter_scoring: true # Whether the gdist_scale and pdist_scale
parameters should assume that goal_distance and path_distance are
expressed in units of meters or cells. Cells are assumed by default
(false).
  occdist_scale:  0.1 #The weighting for how much the controller
should attempt to avoid obstacles. default 0.01
  pdist_scale: 0.75  #     The weighting for how much the
controller should stay close to the path it was given . default 0.6
  gdist_scale: 1.0 #     The weighting for how much the controller
should attempt to reach its local goal, also controls speed
default 0.8

  heading_lookahead: 0.325  #How far to look ahead in meters when
scoring different in-place-rotation trajectories
  heading_scoring: false  #Whether to score based on the robot's
heading to the path or its distance from the path. default false
  heading_scoring_timestep: 0.8   #How far to look ahead in time in
seconds along the simulated trajectory when using heading scoring
(double, default: 0.8)
  dwa: true #Whether to use the Dynamic Window Approach (DWA)_ or
whether to use Trajectory Rollout
  simple_attractor: false
  publish_cost_grid_pc: true

  # Oscillation Prevention Parameters
  oscillation_reset_dist: 0.25 #How far the robot must travel in
meters before oscillation flags are reset (double, default: 0.05)
  escape_reset_dist: 0.1
  escape_reset_theta: 0.1

DWAPlannerROS:
  # Robot configuration parameters
  acc_lim_x: 2.5
  acc_lim_y: 0
  acc_lim_th: 3.2

  max_vel_x: 0.5

  min_vel_x: 0.0
  max_vel_y: 0
  min_vel_y: 0

  max_trans_vel: 0.5
  min_trans_vel: 0.1
  max_rot_vel: 1.0
```

```
  min_rot_vel: 0.2

  # Goal Tolerance Parameters
  yaw_goal_tolerance: 0.1
  xy_goal_tolerance: 0.2
  latch_xy_goal_tolerance: false

  # # Forward Simulation Parameters
  # sim_time: 2.0
  # sim_granularity: 0.02
  # vx_samples: 6
  # vy_samples: 0
  # vtheta_samples: 20
  # penalize_negative_x: true

  # # Trajectory scoring parameters
  # path_distance_bias: 32.0 # The weighting for how much the
controller should stay close to the path it was given
  # goal_distance_bias: 24.0 # The weighting for how much the
controller should attempt to reach its local goal, also controls
speed
  # occdist_scale: 0.01 # The weighting for how much the controller
should attempt to avoid obstacles
  # forward_point_distance: 0.325 # The distance from the center
point of the robot to place an additional scoring point, in meters
  # stop_time_buffer: 0.2  # The amount of time that the robot must
stThe absolute value of the veolicty at which to start scaling the
robot's footprint, in m/sop before a collision in order for a
trajectory to be considered valid in seconds
  # scaling_speed: 0.25 # The absolute value of the veolicty at
which to start scaling the robot's footprint, in m/s
  # max_scaling_factor: 0.2 # The maximum factor to scale the
robot's footprint by

  # # Oscillation Prevention Parameters
  # oscillation_reset_dist: 0.25 #How far the robot must travel in
meters before oscillation flags are reset (double, default: 0.05)
```

El planificador global es el encargado de generar una ruta de navegación sobre el espacio libre considerando la forma y restricciones cinemáticas del robot. El planificador local es el encargado de detectar obstáculos y buscar una ruta alternativa hasta conectar de nuevo con la ruta preestablecida por el planificador global. Para ello hace uso de los sensores implementados.

El siguiente archivo *.launch* permite comenzar la navegación sin mapa precargado.

```
roslaunch sumit_xl_tools start_navigation_without_map.launch
```

De esta forma iniciará la navegación con solo odometría y láser, generando Costmaps o mapa de costes locales y globales en tiempo real gracias a las mediciones del LiDAR. Esto

se podrá usar para planificar la mejor ruta en función de los obstáculos que detecte. Los mapas globales contienen la información que el robot tiene previamente (paredes, objetos estáticos, etc.). Los mapas locales contienen la información que se va generando en tiempo real gracias a los sensores que el robot incorpora.

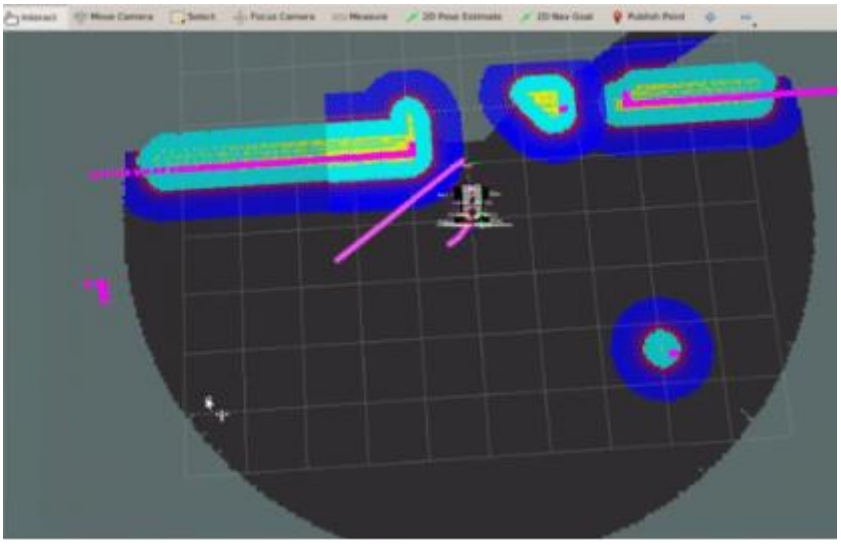

Figura 1. Robot con mapa local representado

Mediante el siguiente archive *.launch*, el robot podrá ir navegando empleado el mapa global precargado e ir generando el mapa local con los obstáculo que se vaya encontrando. De esta forma el robot puede navegar más rápido.

```
roslaunch sumit_xl_tools start_navigation_with_map_v2.launch
```

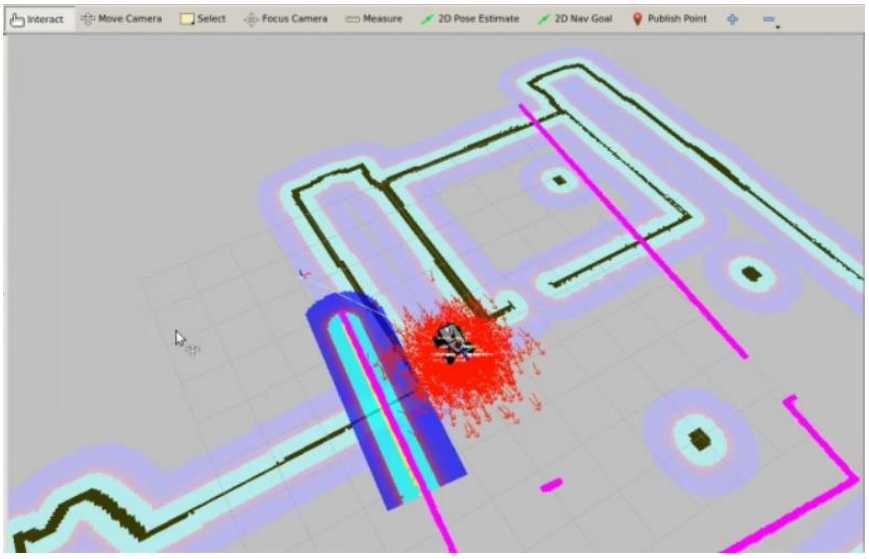

Figura 2. Robot con mapa global y local

Tanto los mapas globales y locales como las mediciones realizadas por los sensores (LiDAR y cámara) pueden ser visualizadas usando la interfaz gráfica Rviz. Rviz es una potente herramienta para poder visualizar en 3D y en tiempo real lo que el robot está viendo y qué está haciendo. Permite suscribirse a los topics que se estén publicando en ROS y mostrarlos en el sistema de coordenadas que se esté empleando.

Para generar un mapa y guardar un mapa se emplea el paquete GMapping. El archivo gmapping.launch tiene la siguiente estructura:

```
<launch>
    <node pkg="gmapping" type="slam_gmapping" name="slam_gmapping"
output="screen">

      <remap from="scan" to ="/hokuyo_base/scan"/>
      <param name="base_frame" value="summit_xl_a_base_footprint"/>
      <param name="odom_frame" value="summit_xl_a_odom"/>
      <!-- Process 1 out of every this many scans (set it to a
higher number to skip more scans)  -->
    <param name="throttle_scans" value="1"/>

    <param name="map_update_interval" value="5.0"/> <!-- default:
5.0 -->

    <!-- The maximum usable range of the laser. A beam is cropped
to this value.  -->
    <param name="maxUrange" value="5.0"/>

    <!-- The maximum range of the sensor. If regions with no
obstacles within the range of the sensor should appear as free
space in the map, set maxUrange < maximum range of the real sensor
<= maxRange -->
    <param name="maxRange" value="10.0"/>
      <param name="sigma" value="0.05"/>
      <param name="kernelSize" value="1"/>
      <param name="lstep" value="0.05"/>
      <param name="astep" value="0.05"/>
      <param name="iterations" value="5"/>
      <param name="lsigma" value="0.075"/>
      <param name="ogain" value="3.0"/>
      <param name="lskip" value="0"/>
      <param name="srr" value="0.1"/>
      <param name="srt" value="0.2"/>
      <param name="str" value="0.1"/>
      <param name="stt" value="0.2"/>
      <param name="linearUpdate" value="0.2"/>
      <param name="angularUpdate" value="0.1"/>
      <param name="temporalUpdate" value="3.0"/>
      <param name="resampleThreshold" value="0.5"/>
      <param name="particles" value="100"/>
      <param name="xmin" value="-50.0"/>
      <param name="ymin" value="-50.0"/>
      <param name="xmax" value="50.0"/>
      <param name="ymax" value="50.0"/>
```

```
      <param name="delta" value="0.05"/>
      <param name="llsamplerange" value="0.01"/>
      <param name="llsamplestep" value="0.01"/>
      <param name="lasamplerange" value="0.005"/>
      <param name="lasamplestep" value="0.005"/>
    </node>
</launch>
```

Para guardar el mapa se ejecutan las siguientes instrucciones en un terminal del robot:

```
roscd my_summit_xl_tools
mkdir maps
cd maps
rosrun map_server map_saver -f mymap
```

Esto creará *mymap.pgm y mymap.yaml*. Utilizará la imagen *.pgm* para limpiar el mapa generado y el *.yaml* es el archivo directamente utilizado por el planificador después. El mapa podrá ser usado posteriormente lanzando el archivo *start_navigation_with_map.launch.*

```
<launch>

    <!-- Run the map server -->
    <arg name="map_file" default="$(find
my_summit_xl_tools)/maps/mymap.yaml"/>
    <node name="map_server" pkg="map_server" type="map_server"
args="$(arg map_file)" />

    <!--- Run AMCL -->
    <include file="$(find my_summit_xl_tools)/launch/amcl.launch"
/>

    <!--- Run Move Base -->
    <include file="$(find
my_summit_xl_tools)/launch/move_base_map.launch" />

</launch>
```

Donde *map_server* es el nodo que publica el mapa guardado, *Amcl* es el nodo que localiza el robot en función de las lecturas de mapas y *move_base* tiene la configuración para moverse con un mapa.

Para crear una ruta determinada y que el robot pueda ir desde el origen al destino pasando por los puntos determinados, se requiere de la creación de un servicio que registre dichas ubicaciones e interactúe con la Navigation Stack. El programa presenta la siguiente estructura:

```
#!/usr/bin/env python

import threading
import rospy
import actionlib

from smach import State,StateMachine
from move_base_msgs.msg import MoveBaseAction, MoveBaseGoal
from geometry_msgs.msg import PoseWithCovarianceStamped, PoseArray
from std_msgs.msg import Empty

waypoints = []

class FollowPath(State):
    def __init__(self):
        State.__init__(self, outcomes=['success'],
input_keys=['waypoints'])
        self.frame_id = rospy.get_param('~goal_frame_id','map')
        # Get a move_base action client
        self.client = actionlib.SimpleActionClient('move_base',
MoveBaseAction)
        rospy.loginfo('Connecting to move_base...')
        self.client.wait_for_server()
        rospy.loginfo('Connected to move_base.')

    def execute(self, userdata):
        global waypoints
        # Execute waypoints each in sequence
        for waypoint in waypoints:
            # Break if preempted
            if waypoints == []:
                rospy.loginfo('The waypoint queue has been reset.')
                break
            # Otherwise publish next waypoint as goal
            goal = MoveBaseGoal()
            goal.target_pose.header.frame_id = self.frame_id
            goal.target_pose.pose.position =
waypoint.pose.pose.position
            goal.target_pose.pose.orientation =
waypoint.pose.pose.orientation
            rospy.loginfo('Executing move_base goal to position
(x,y): %s, %s' %
                    (waypoint.pose.pose.position.x,
waypoint.pose.pose.position.y))
            rospy.loginfo("To cancel the goal: 'rostopic pub -1
/move_base/cancel actionlib_msgs/GoalID -- {}'")
            self.client.send_goal(goal)
            self.client.wait_for_result()
        return 'success'

def convert_PoseWithCovArray_to_PoseArray(waypoints):
    """Used to publish waypoints as pose array so that you can see
them in rviz, etc."""
    poses = PoseArray()
```

```
    poses.header.frame_id = 'map'
    poses.poses = [pose.pose.pose for pose in waypoints]
    return poses

class GetPath(State):
    def __init__(self):
        State.__init__(self, outcomes=['success'],
input_keys=['waypoints'], output_keys=['waypoints'])
        # Create publsher to publish waypoints as pose array so
that you can see them in rviz, etc.
        self.poseArray_publisher = rospy.Publisher('/waypoints',
PoseArray, queue_size=1)
        self._custom_waypointstopic =
rospy.get_param('~custom_waypointstopic','/my_waypoints_list')
        # Start thread to listen for reset messages to clear the
waypoint queue
        def wait_for_path_reset():
            """thread worker function"""
            global waypoints
            while not rospy.is_shutdown():
                data = rospy.wait_for_message('/path_reset', Empty)
                rospy.loginfo('Recieved path RESET message')
                self.initialize_path_queue()
                rospy.sleep(3) # Wait 3 seconds because `rostopic
echo` latches
                               # for three seconds and
wait_for_message() in a
                               # loop will see it again.
        reset_thread = threading.Thread(target=wait_for_path_reset)
        reset_thread.start()

    def initialize_path_queue(self):
        global waypoints
        waypoints = [] # the waypoint queue
        # publish empty waypoint queue as pose array so that you
can see them the change in rviz, etc.

self.poseArray_publisher.publish(convert_PoseWithCovArray_to_PoseAr
ray(waypoints))

    def execute(self, userdata):
        global waypoints
        self.initialize_path_queue()
        self.path_ready = False

        # Start thread to listen for when the path is ready (this
function will end then)
        def wait_for_path_ready():
            """thread worker function"""
            data = rospy.wait_for_message('/path_ready', Empty)
            rospy.loginfo('Recieved path READY message')
            self.path_ready = True
        ready_thread = threading.Thread(target=wait_for_path_ready)
        ready_thread.start()
```

```
topic = self._custom_waypointstopic
        rospy.loginfo("Waiting to recieve waypoints via Pose msg on
topic %s" % topic)
        rospy.loginfo("To start following waypoints: 'rostopic pub
/path_ready std_msgs/Empty -1'")

        # Wait for published waypoints
        while not self.path_ready:
            try:
                pose = rospy.wait_for_message(topic,
PoseWithCovarianceStamped, timeout=1)
            except rospy.ROSException as e:
                if 'timeout exceeded' in e.message:
                    continue  # no new waypoint within timeout,
looping...
                else:
                    raise e
            rospy.loginfo("Recieved new waypoint")
            waypoints.append(pose)
            # publish waypoint queue as pose array so that you can
see them in rviz, etc.

self.poseArray_publisher.publish(convert_PoseWithCovArray_to_PoseAr
ray(waypoints))

        # Path is ready! return success and move on to the next
state (FOLLOW_PATH)
        return 'success'

class PathComplete(State):
    def __init__(self):
        State.__init__(self, outcomes=['success'])

    def execute(self, userdata):
        rospy.loginfo('##############################')
        rospy.loginfo('#####  REACHED FINISH GATE  #####')
        rospy.loginfo('##############################')
        return 'success'

def main():
    rospy.init_node('custom_follow_waypoints')

    sm = StateMachine(outcomes=['success'])

    with sm:
        StateMachine.add('GET_PATH', GetPath(),
                         transitions={'success':'FOLLOW_PATH'},
                         remapping={'waypoints':'waypoints'})
        StateMachine.add('FOLLOW_PATH', FollowPath(),
                         transitions={'success':'PATH_COMPLETE'},
                         remapping={'waypoints':'waypoints'})
        StateMachine.add('PATH_COMPLETE', PathComplete(),
                         transitions={'success':'GET_PATH'})
```

```
 outcome = sm.execute()

if __name__ == "__main__":
    main()
```

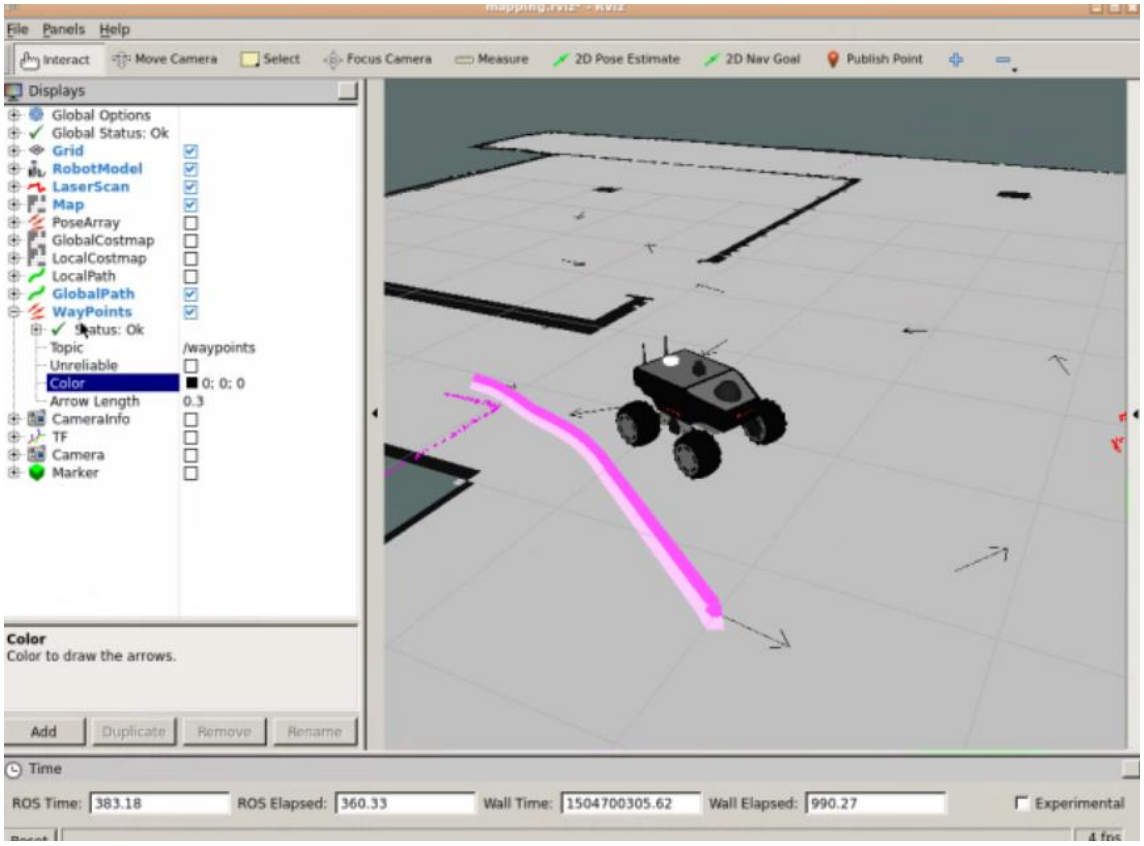

Figura 3. Ejemplo de robot siguiendo una ruta determinada a través de Rviz

Los programas anteriores están centrados en la navegación sin emplear el GPS, lo que requeriría realizar o disponer de un mapa previamente de la parcela para poder navegar sorteando obstáculos. A continuación, se detallará cómo hacerlo basándose en las coordenadas GPS. En este caso es necesario prescindir del nodo Amcl, encargado de conectar el mapa y la odometría y se empleará el paquete robot_localization que fusiona datos de sensores, GPS, visión, IMU, etc. mediante filtros Kalman para una mejor localización. Para esto se empleará el programa siguiente:

start_navigation_with_gps_ekf.launch

```
<launch>
    <include file="$(find
my_summit_xl_tools)/launch/start_navsat.launch" />
```

```
    <!-- Run the ekf for map to odom config -->
    <node pkg="robot_localization" type="ekf_localization_node" name="ekf_localization_with_gps">
    <rosparam command="load" file="$(find my_summit_xl_tools)/config/robot_localization_with_gps.yaml" />
    </node>

    <!-- Run the map server -->
    <include file="$(find my_summit_xl_tools)/launch/start_map_server.launch" />

    <!--- Run Move Base -->
    <include file="$(find my_summit_xl_tools)/launch/move_base_map.launch" />

    <!-- Start RVIZ for Localization -->
    <include file="$(find my_summit_xl_tools)/launch/view_robot.launch">
        <arg name="config" value="navigation_map" />
    </include>

</launch>
```

teniendo *move_base_map.launch* la siguiente configuración:

```
<?xml version="1.0"?>
<!-- NEW SUMMIT XL NAVIGATION -->
<launch>

    <arg name="no_static_map" default="false"/>

    <arg name="base_global_planner" default="navfn/NavfnROS"/>
    <arg name="base_local_planner" default="dwa_local_planner/DWAPlannerROS"/>
    <!-- <arg name="base_local_planner" default="base_local_planner/TrajectoryPlannerROS"/> -->

    <node pkg="move_base" type="move_base" respawn="false" name="move_base" output="screen">

        <param name="base_global_planner" value="$(arg base_global_planner)"/>
        <param name="base_local_planner" value="$(arg base_local_planner)"/>
        <rosparam file="$(find my_summit_xl_tools)/config/planner.yaml" command="load"/>

        <!-- observation sources located in costmap_common.yaml -->
        <rosparam file="$(find my_summit_xl_tools)/config/costmap_common.yaml" command="load" ns="global_costmap" />
        <rosparam file="$(find my_summit_xl_tools)/config/costmap_common.yaml" command="load" ns="local_costmap" />
```

```
        <!-- local costmap, needs size -->
        <rosparam file="$(find my_summit_xl_tools)/config/costmap_local.yaml" command="load" ns="local_costmap" />
        <param name="local_costmap/width" value="5.0"/>
        <param name="local_costmap/height" value="5.0"/>

        <!-- static global costmap, static map provides size -->




  <rosparam file="$(find my_summit_xl_tools)/config/costmap_global_static.yaml" command="load" ns="global_costmap" unless="$(arg no_static_map)"/>

        <!-- global costmap with laser, for odom_navigation_demo -->
        <rosparam file="$(find my_summit_xl_tools)/config/costmap_global_laser.yaml" command="load" ns="global_costmap" if="$(arg no_static_map)"/>
        <param name="global_costmap/width" value="100.0" if="$(arg no_static_map)"/>
        <param name="global_costmap/height" value="100.0" if="$(arg no_static_map)"/>

        <remap from="/cmd_vel" to="/summit_xl_control/cmd_vel" />
    </node>

</launch>
```

Por otro lado, también se usa el nodo *navsat_transform_node* que permite convertir los datos GPS (latitud y longitud) en coordenadas XY representables en el espacio. La estructura del programa que lanza este nodo es la siguiente:

start_navsat.launch

```
<launch>
 <!-- -->
  <node pkg="robot_localization" type="navsat_transform_node" name="navsat_transform_node" respawn="true">

    <param name="magnetic_declination_radians" value="0"/>
    <param name="yaw_offset" value="0"/>
    <param name="zero_altitude" value="true"/>
```

```
 <param
name="broadcast_utm_transform"
value="false"/>
 <param
name="publish_filtered_gps" value="false"/>


 <param name="use_odometry_yaw" value="false"/>
 <param name="wait_for_datum" value="false"/>

    <remap from="/imu/data" to="/imu/data" />
    <remap from="/gps/fix" to="/gps/fix" />
    <remap from="/odometry/filtered"
to="/robotnik_base_control/odom" />

  </node>

</launch>
```

La configuración de los diferentes sensores incorporados y empleados para la navegación por GPS se realiza a través del archivo de configuración *robot_localization_with_gps.yaml*. Este archivo presenta la siguiente estructura:

```
#Configuation for robot odometry EKF
#
frequency: 50
odom0: /robotnik_base_control/odom
odom0_config: [false, false, false,
               false, false, false,
               true, true, true,
               false, false, true,
               false, false, false]
odom0_differential: true

imu0: /imu/data
imu0_config: [false, false, false,
              true, true, true,
              false, false, false,
              true, true, true,
              false, false, false]
imu0_differential: false

odom1: /odometry/gps
odom1_config: [false, false, false,
               false, false, false,
               true, true, true,
               false, false, true,
               false, false, false]
odom1_differential: false

odom_frame: summit_xl_a_odom
base_link_frame: summit_xl_a_base_footprint
world_frame: map
map_frame: map
```

```
process_noise_covariance": [0.05,
0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
0,    0,    0,
                                                0,    0.05, 0,    0,
0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
                                                0,    0,    0.06, 0,
0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
                                                0,    0,    0,
0.03, 0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
0,
                                                0,    0,    0,    0,
0.03, 0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
                                                0,    0,    0,    0,
0,    0.06, 0,    0,    0,    0,    0,    0,    0,    0,    0,
                                                0,    0,    0,    0,
0,    0,    0.025, 0,    0,    0,    0,    0,    0,    0,    0,
                                                0,    0,    0,    0,
0,    0,    0,    0.025, 0,    0,    0,    0,    0,    0,    0,
                                                0,    0,    0,    0,
0,    0,    0,    0,    0.04, 0,    0,    0,    0,    0,    0,
                                                0,    0,    0,    0,
0,    0,    0,    0,    0,    0.01, 0,    0,    0,    0,    0,
                                                0,    0,    0,    0,
0,    0,    0,    0,    0,    0,    0.01, 0,    0,    0,    0,
                                                0,    0,    0,    0,
0,    0,    0,    0,    0,    0,    0,    0.02, 0,    0,    0,
                                                0,    0,    0,    0,
0,    0,    0,    0,    0,    0,    0,    0,    0.01, 0,    0,
                                                0,    0,    0,    0,
0,    0,    0,    0,    0,    0,    0,    0,    0,    0.01, 0,
                                                0,    0,    0,    0,
0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
0.015]


initial_estimate_covariance: [1e-9, 0,    0,    0,    0,    0,
0,    0,    0,    0,    0,    0,    0,    0,    0,
                                                0,    1e-9,
0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
0,    0,
                                                0,    0,
1e-9, 0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
0,    0,
                                                0,    0,
0,    1e-9, 0,    0,    0,    0,    0,    0,    0,    0,    0,
0,    0,
                                                0,    0,
0,    0,    1e-9, 0,    0,    0,    0,    0,    0,    0,    0,
0,    0,
                                                0,    0,
0,    0,    0,    1e-9, 0,    0,    0,    0,    0,    0,    0,
0,    0,
                                                0,    0,
0,    0,    0,    0,    1e-9, 0,    0,    0,    0,    0,    0,
0,    0,
```

```
0,     0,     0,     0,     0,     0,     0,     1e-9, 0,     0,     0,
0,     0,     0,     0,
                                                          0,     0,
0,     0,     0,     0,     0,     0,     1e-9, 0,     0,     0,     0,
0,     0,
                                                          0,     0,
0,     0,     0,     0,     0,     0,     0,     1e-9,  0,     0,     0,
0,     0,
                                                          0,     0,
0,     0,     0,     0,     0,     0,     0,     0,     1e-9,  0,     0,
0,     0,
                                                          0,     0,
0,     0,     0,     0,     0,     0,     0,     0,     0,     1e-9,  0,
0,     0,
                                                          0,     0,
0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     1e-
9, 0,     0,
                                                          0,     0,
0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
1e-9, 0,
                                                          0,     0,
0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
0,     1e-9]
```

Al disponer de GPS, es posible representar el robot moviéndose por una base cartográfica suscribiéndose al topic /gps/fix. De esta forma se puede conocer su posición real (Figura 4).

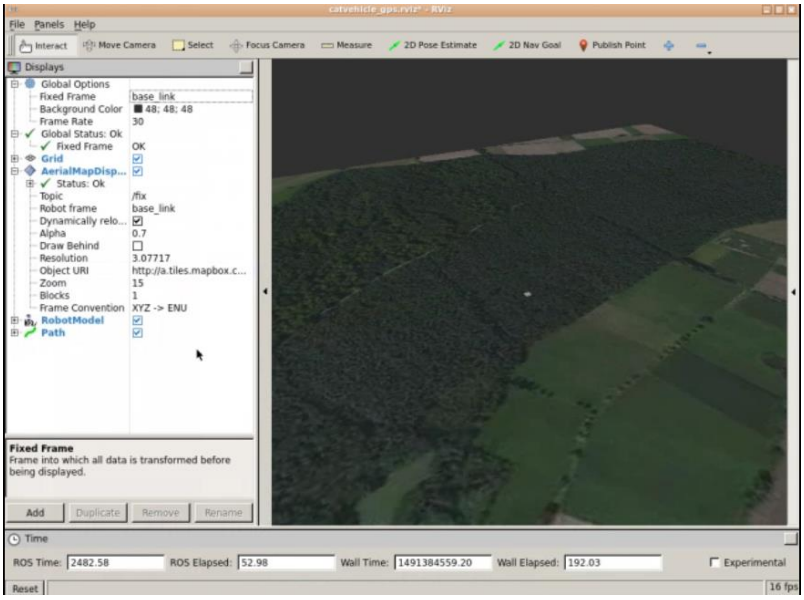

Figura 4. Representación mediante base cartográfica

Para enviar posición GPS al robot por las que este deba moverse es necesario transformarlas a coordenadas XYZ desde el origen de la ubicación GPS. Para ello se usa

el paquete *geonav_transform*. EL programa desarrollado para convertir coordenadas GPS (latitud y longitud) a coordenadas XYZ es el siguiente (*gps_to_xyz.py*).

```
#!/usr/bin/env python

# Import geonav tranformation module
import geonav_transform.geonav_conversions as gc
reload(gc)
# Import AlvinXY transformation module
import alvinxy.alvinxy as axy
reload(axy)
import rospy
import tf
from nav_msgs.msg import Odometry


def get_xy_based_on_lat_long(lat,lon, name):
    # Define a local orgin, latitude and longitude in decimal
degrees
    # GPS Origin
    olat = 49.9
    olon = 8.9

    xg2, yg2 = gc.ll2xy(lat,lon,olat,olon)
    utmy, utmx, utmzone = gc.LLtoUTM(lat,lon)
    xa,ya = axy.ll2xy(lat,lon,olat,olon)

    rospy.loginfo("#########  "+name+"  ###########")
    rospy.loginfo("LAT COORDINATES ==>"+str(lat)+","+str(lon))
    rospy.loginfo("COORDINATES XYZ ==>"+str(xg2)+","+str(yg2))
    rospy.loginfo("COORDINATES AXY==>"+str(xa)+","+str(ya))
    rospy.loginfo("COORDINATES UTM==>"+str(utmx)+","+str(utmy))

    return xg2, yg2

if __name__ == '__main__':
    rospy.init_node('gps_to_xyz_node')
    xg2, yg2 = get_xy_based_on_lat_long(lat=49.9,lon=8.9,
name="MAP")
    xg2, yg2 = get_xy_based_on_lat_long(lat=50.9,lon=8.9,
name="MAP")
```

3. ALGORITMOS DE DETECCIÓN DE PERSONAS Y OBJETOS

Para la detección de personas se usa habitualmente un detector de piernas mediante láser, aunque también se puede realizar mediante una cámara y procesamiento de imágenes con OpenCV. En este caso se usará el paquete *leg-detector.* El archivo .launch tendrá la siguiente estructura:

start_legdetector.launch

```
<launch>
    <arg name="scan" default="/hokuyo_base/scan" />
    <arg name="machine" default="localhost" />
    <arg name="user" default="" />
    <!-- Leg Detector -->
    <node pkg="leg_detector" type="leg_detector"
name="leg_detector" args="scan:=$(arg scan) $(find
leg_detector)/config/trained_leg_detector.yaml" respawn="true"
output="screen">
        <param name="fixed_frame" type="string" value="map" />
    </node>

</launch>
```

Este programa proporcionará información sobre la posición de la persona y la fiabilidad de la detección. Además, la detección se podrá configurar y representar mediante el visualizador Rviz y el programa siguiente:

leg_detector_marker_publisher.py

```
#!/usr/bin/env python
import rospy
from people_msgs.msg import PositionMeasurementArray


#!/usr/bin/env python

import rospy
from visualization_msgs.msg import Marker
from geometry_msgs.msg import Point


class MarkerBasics(object):

    def __init__(self):
        self.marker_objectlisher =
rospy.Publisher('/marker_person_detected_leg', Marker,
queue_size=1)
        self.rate = rospy.Rate(1)
        self.init_marker(index=0,z_val=0)

    def init_marker(self,index=0, z_val=0):
        self.marker_object = Marker()
        self.marker_object.header.frame_id = "/map"
        self.marker_object.header.stamp    = rospy.get_rostime()
        self.marker_object.ns = "summit_xl"
        self.marker_object.id = index
        self.marker_object.type = Marker.MESH_RESOURCE
        self.marker_object.action = Marker.ADD

        my_point = Point()
```

```
self.marker_object.pose.position = my_point

        self.marker_object.pose.orientation.x = 0
        self.marker_object.pose.orientation.y = 0
        self.marker_object.pose.orientation.z = 1.0
        self.marker_object.pose.orientation.w = 1.0
        self.marker_object.scale.x = 1.0
        self.marker_object.scale.y = 1.0
        self.marker_object.scale.z = 1.0

        self.marker_object.color.r = 0.0
        self.marker_object.color.g = 0.0
        self.marker_object.color.b = 0.0
        # This has to be otherwise it will be transparent
        self.marker_object.color.a = 0.0
        self.marker_object.mesh_resource =
"package://person_sim/models/person_standing/meshes/standingv2.dae"
        self.marker_object.mesh_use_embedded_materials = True
        # If we want it for ever, 0, otherwise seconds before
desapearing
        self.marker_object.lifetime = rospy.Duration(0)

    def update_position(self,position, reliability):

        self.marker_object.pose.position = position

        self.marker_objectlisher.publish(self.marker_object)

"""
### PositionMeasurementArray

std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
people_msgs/PositionMeasurement[] people
  std_msgs/Header header
    uint32 seq
    time stamp
    string frame_id
  string name
  string object_id
  geometry_msgs/Point pos
    float64 x
    float64 y
    float64 z
  float64 reliability
  float64[9] covariance
  byte initialization
float32[] cooccurrence
"""
class LegDetector(object):
    def __init__(self):
```

```
rospy.Subscriber("/leg_tracker_measurements",
PositionMeasurementArray, self.leg_detect_callback)

        self.markerbasics_object = MarkerBasics()



    def leg_detect_callback(self,data):
        for people_object in data.people:

self.markerbasics_object.update_position(people_object.pos,
people_object.reliability)

    def start_loop(self):
        # spin() simply keeps python from exiting until this node
is stopped
        rospy.spin()

if __name__ == '__main__':
    rospy.init_node('leg_detections_listener_node', anonymous=True)
    legdetector_object = LegDetector()
    legdetector_object.start_loop()
```

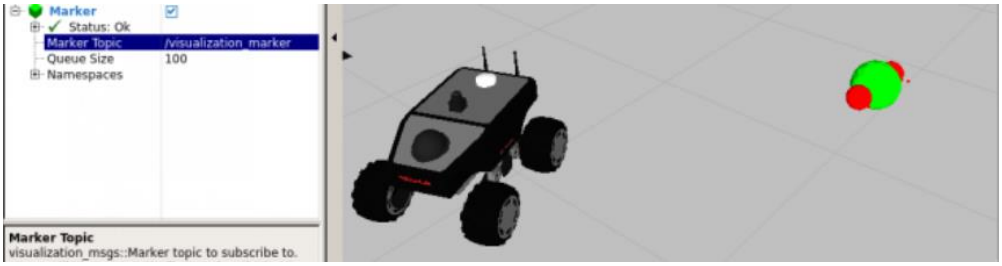

Figura 5. Representación en Rviz de la detección de una persona

Para detectar objetos en ROS mediante imágenes de una cámara, por ejemplo, la ZED2 de la solución robótica propuesta, se emplea YOLO (You only look once). YOLO es un sistema de detección de objetos en tiempo real que se implementa en la GPU y la CPU y que depende de OpenCV. El modelo pre-entrenado de la red neuronal convolucional es capaz de detectar clases pre-entrenadas o también crear una red con sus propios objetos de detección. El archivo *.launch* tendrá la siguiente estructura:

Yolo_v3.launch

```
<?xml version="1.0" encoding="utf-8"?>

<launch>

  <!-- Use YOLOv3 -->
  <arg name="network_param_file"         default="$(find
darknet_ros)/config/yolov3.yaml"/>
  <arg name="image" default="camera/rgb/image_raw" />


  <!-- Include main launch file -->
  <include file="$(find darknet_ros)/launch/darknet_ros.launch">
    <arg name="network_param_file"    value="$(arg
network_param_file)"/>
    <arg name="image" value="$(arg image)" />
  </include>
```


Interreg
España - Portugal
Fondo Europeo de Desarrollo Regional
UNION EUROPEA
COMPETITIVIDAD
HIBA
HUB IBERIA AGROTECH
Junta de Andalucía
Consejería de Agricultura, Ganadería, Pesca y Desarrollo Sostenible
UNIVERSIDAD D CÓRDOBA
Junta de Andalucía
Consejería de Transformación Económica, Industria, Conocimiento y Universidades
AXENCIA GALEGA DE INNOVACIÓN
CATEC
ADVANCED CENTER FOR AEROSPACE TECHNOLOGIES
FADA
ANDALUSIAN FOUNDATION FOR AEROSPACE DEVELOPMENT
INTEC
FUNDACIÓN EUROPEA PARA LA INNOVACIÓN Y APLICACIÓN DE LA TECNOLOGÍA
Finnova
GOBIERNO DE ESPAÑA
MINISTERIO DE DEFENSA
INTA
INSTITUTO NACIONAL DE TÉCNICA AEROESPACIAL
DiH DATAlife
FUNDECYT PCTEX
UNIVERSIDADE DE ÉVORA
cooperativas agro-alimentarias Andalucía
iniav
Instituto Nacional de Investigação Agrária e Veterinária, I.P.
UAlg FCT
UNIVERSIDADE DO ALGARVE
FACULDADE DE CIÊNCIAS E TECNOLOGIA
Galileo Galilei
Seguimos avanzando
IPBeja
INSTITUTO POLITÉCNICO DE BEJA
SMART FARM COLAB
LABORATÓRIO COLABORATIVO PARA A INOVAÇÃO DIGITAL NA AGRICULTURA
INL
INTERNATIONAL IBERIAN NANOTECHNOLOGY LABORATORY
CONSULAI
www.consulai.com