
# The best follower: a robotic agent

**By:**

Jesús de Atocha Anaya Correa A01703175

June 06, 2022

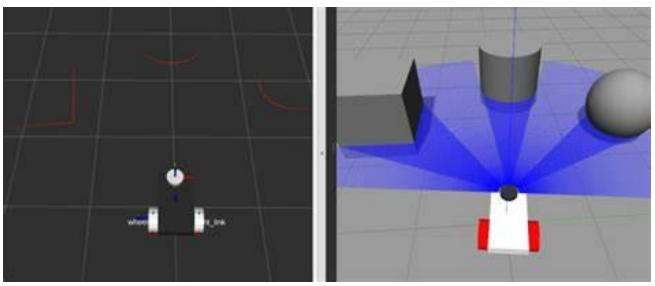
## ABSTRACT

There is an increasing lack of workers in agricultural jobs, principally because the hard work required to accomplish and the extreme climate conditions specially in greenhouses where the humidity and heat is very high. This project intents to make easier and faster the harvesting task of tomatoes with an robotic goal agent which follows the worker using GPS-RTK and a obstacle avoidance algorithm. This project is just the start to a bigger implementation of a hole autonomous harvesting robot in the future.


## INTRODUCTION

Nowadays robots play an important role in the automation industry. They increase the productivity and efficiency of hard and repetitive tasks like manufacturing, assembling, or manipulating piece and objects precisely.
Agriculture is one of the most challenging fields when talking about automation because there are more variables to take into account than in a boring factory where all the variables and parameters are completely known and rarely there are changes once the production line is settled. Furthermore, the tasks required in agriculture for example in greenhouses still represent a lot of physical work. So in order to reduce the amount of kilograms that a worker loads and to let the worker with free hands to perform its different tasks on the plants a robotic rational agent will be implemented which will be following the worker by following the latitude and longitude coordinate of the worker GPS (RTK). But what is a rational agent? According with Bansall(2022) a rational agent could be anything that makes decisions by its own and acts upon an environment with its actuators and also sense the environment with its sensors. For example, a person, firm, machine, or software. The agents act in their environment. The environment may contain other agents. A robotic agent has cameras and infrared range finders, IMUs, LiDAR, which act as sensors and motors, pistons, gripers acting as actuators.

A LiDAR (Light Detection and Ranging) is a sensor which send out pulses of light to the world and has a scanner which once the pulses has bounce from objects it receives and record the time delay between light pulse transmission and reception to calculate the distance between the sensor and the objects Fig1.



(Robotisim, n.d.)
Fig. 1. Visualising the LiDAR with RVIZ and gazebo.

Another sensor that the robot presented in this project will use is a Inertial Measurement Unit (IMU), which is made of (VECTORNAV, n.d.):
Gyroscopes: providing a measure angular rate.

Accelerometers: providing a measure specific force/acceleration.
Magnetometers (optional): measurement of the magnetic field surrounding the system.
The information required to take from the IMU sensor is the current orientation of the robot which is given by the Gyroscope.

The RTK is short for real time kinematics which basically is a GPS receiver that takes normal signals from the Global Navigation Satellite Systems and makes a correction stream to reduce the error carried by the normal GPS which is approximately 10-20 meters to an 1cm positional accuracy (NATE, n.d.).

The principal software used to control and simulate the robot is ROS (Robot Operating System) and Gazebo which only run in ubuntu. ROS as its name suggests is a set of open-source software libraries and tools that help built and control robot applications, has drivers of state-of-the-art-algorithms and powerful and flexible developer tools to build any robotic application that can be imagined (ROS, 2021). And Gazebo which is used to simulate the robot and environment in 3D to test the performance of the robot of this project called UMA (Modular Autonomous Unit) shown in figure 2. This simulator is particularly suitable for testing object-avoidance so that is the principal reason it was chosen.

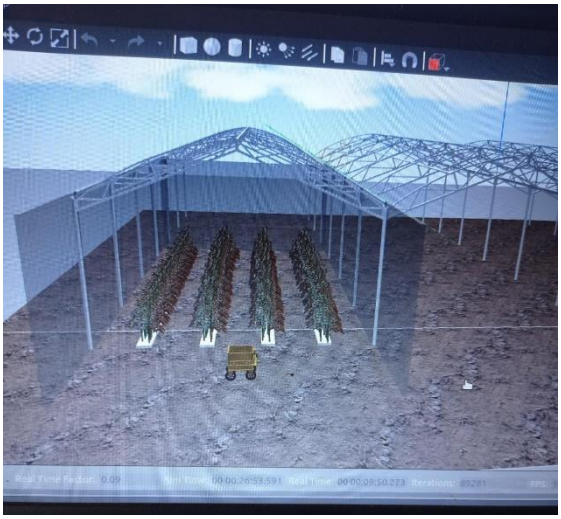


Fig. 2. UMA in a greenhouse simulated in gazebo.

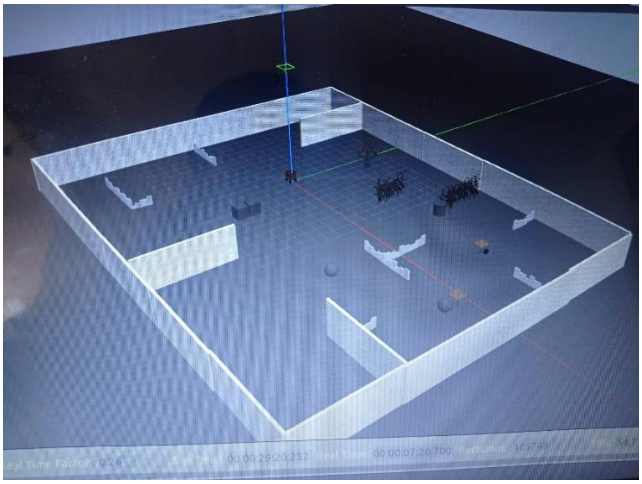


Fig. 3. UMA in a simulated room with objects

## State of the art

One of the state-of-the-art robotic agents in agriculture is the one shown in figure 4. Called Sweeper, this agent is a robot for harvesting autonomously sweet pepper fruit in greenhouses, is a learning agent which based in its previous knowledge and feedback harvest only the peppers that are ready to be harvested. One of the important aspects of state of the art of this robot is the precise control to travel autonomously through the

greenhouse while harvesting and also the computer vision algorithm to detect the peppers and select which are ready to be harvested. They also use ROS to control some parts of the robot and the communication between the base mobile platform and the arm. His performance still is not as fast as a human worker so there is an area of opportunity to future applications (B. Arad, et al. 2020).


(B. Arad, et al., 2020).
Fig. 4. Harvesting robot Sweeper.

## AGENT PROPOSED

The robotic agent proposed to this application is a goal-based agent due to the nature of the task to perform. The goal of the robot is to follow a person or any other agent which has a gps-rtk with him which will be the one sending the goal position (coordinates of latitude and longitude) to the robot to create and error in the differences of distances and make it move until reaches the goal coordinates within a 50cm of proximity, so the robot does not hit the person or the other agent.

This application is the beginning of a hole autonomous tomatoes harvesting robot. In this stage the pretended outcome is to make the robot follow a person so that the person can be harvesting without the need of been carrying a box of tomatoes or pushing a pallet truck or wheelbarrow which are the more commonly used tool nowadays.

The first thing done was to set the workspace environment in which the robot will be performing, so two different environments were created in the simulator gazebo. The first one shown in figure 3 which is a room with different walls and objects inside. And the second one which is a greenhouse which is the final desired application of this agent.
The robotic agent called UMA is built of sensors and actuators.

Its sensors are a 360° LiDAR, an IMU and a GPS-RTK. The LiDAR is used to measure the distance between the robot and the objects in the environment and with that information avoid colliding or hit objects while travelling to the goal position coordinates. The IMU is required to measure the orientation of the robot to know to which side to turn according with the goal position. The sensor GPS-RTK is used to know the coordinates of latitude and longitude in real time of the robot so they can be compared with the goal coordinates which also are send by a GPS-RTK but is hold by the person to follow.

The actuators of the robot basically are 4 engines which let the robot move its wheels to travel around the simulated world.

# NAVIGATION USING A GOAL COORDINATE.

The gps navigation.py is a node which subscribes to different topics to get the information of the sensors and also this node pushes to a cmd operator to make the motors move.
The first thing done was import the libraries and the specific type of ROS messages which are required to read the sensors and to move the actuators.

```
#!/usr/bin/env python
import rospy
import math    #Para poder usar la tangente inversa, abs()...
from sensor_msgs.msg import NavSatFix #Importar tipo de mensaje que usa el GPS
from sensor_msgs.msg import Imu #Importar tipo de mensaje IMU
from tf.transformations import euler_from_quaternion #Libreria para tranformar los datos del IMU de cuaterniones a angulos de Euler
from nav2d_operator.msg import cmd #Importar tipo de mensaje a publicar a operator
```

In this part the subscription is made to the different publishers.
By subscribing to the fix and fixgoal publisher a message is extracted “self.gps_cb” and “self.gps_goal_cb” which are the messages that contain the coordinates of the robot and the goal respectively.
Also this node subscribes to the publisher imu  from which the orientation of the robot is extracted from the message “self.get_rotation” which is the orientation of the robot in a quaternions form.

This node also publish values in the cmd topic which is the responsible of sending the signals to the motors to be moved and also those values are controlled by the topic obstacle avoidance.

```
# Subscriber para leer el valor del gps del UMA, gps del goal, valor del IMU
rospy.Subscriber("fix", NavSatFix, self.gps_cb)
rospy.Subscriber("fixgoal", NavSatFix, self.gps_goal_cb)
rospy.Subscriber('imu', Imu, self.get_rotation)
####  PUBLISHERS ###
#Publica velocidad a cmd el cual recibe operator para la evasion de obstaculos
self.cmd_pub=rospy.Publisher("cmd", cmd, queue_size = 1)
### CONSTANTS ###
self.robot_vel=cmd()
self.robot_vel.Velocity=0
self.robot_vel.Turn=0
r = rospy.Rate(10) #10 Hz
self.dif_latitude=0.0
self.dif_longitude=0.0
self.latitude_robot=0.0
self.longitude_robot=0.0
self.latitude_goal=0.0
self.longitude_goal=0.0
```

The core part of this node is the computation of the difference in angle between the goal position according with the orientation of the robot. This value of the difference in angle is taken to make the robot turn either clockwise or counter clockwise depending on the nearest turn. Also the linear distance between the goal and robot position is taken to move the motors forward to get to the desired position.

```
#Calculo del angulo del goal respecto al UMA
#Angulos de cuadrantes 1 y 2 respecto a x. En radianes de 0 a Pi (0 -> 180)
#Angulos de cuadrantes 2 y 4 respecto a x. En radianes de 0 a -Pi (0 -> -180)
angle_goal = -1* math.atan2(self.dif_longitude, (self.dif_latitude + 0.0000000000000000001) )

## Movimiento de motores para llegar al objetivo
kdist = 25000 #La constante por la que se multiplica la diferencia de distancia (en terminos de lat y long) entre UMA y goal
kang = -0.20  #Para dejar los valores de velocidad angular en rango de -1 a 1 que son los que lee Turn en cmd
vmax = 1.0    #Para dejar los valores de velocidad lineal en terminos de -1 a 1 que son los que lee Velocity en cmd
latlong = 0.00001317690754 #1m se representa como 0.00001317690754 en terminos de  latitude y longitude aproximadamente
#Calculo de la diferencia de angulo del goal respecto al frente del UMA usando yaw valor generado por el IMU del UMA.
dif_angle = angle_goal - yaw
## Calculo de distancia al goal
dist_to_goal = ((self.dif_latitude ** 2) + (self.dif_longitude ** 2)) ** 0.5
```

In most of the cases the turn value get from the difference in angle is correct to make the shorter turn to the object but there are some special cases where the difference in angle indicates to make a larger turn so those cases are processed differently by correcting the difference by adding a constant to make sure it will always work properly.

```
#Evaluacion de casos especiales en donde la diferencia de angulo no corresponde al valor necesario para girar hacia el goal.
if (angle_goal < -0.07 and angle_goal > -1.50 and yaw > 2.355):
    self.robot_vel.Velocity = (self.robot_vel.Velocity * 0.6) + 0.1
    self.robot_vel.Turn =  kang * (dif_angle + 6.28)
elif (angle_goal < -1.64 and yaw > 0.785):
    self.robot_vel.Velocity = (self.robot_vel.Velocity * 0.6) + 0.1
    self.robot_vel.Turn =  kang * (dif_angle + 6.28)
```

If the position of the goal and the orientation of the robot are not in a

special case the values of the difference of angle and distance are taken like are and just are multiplied by a constant to let the values in the range required by the cmd topic.
And finally if the robot has reached its desired goal it stops the motors when is at a distance of 50cm from its goal position.

```
else:
    self.robot_vel.Velocity = (self.robot_vel.Velocity * 0.6) + 0.1
    self.robot_vel.Turn =  self.robot_vel.Turn =  kang * dif_angle
#Si la velocidad angular de UMA es pequeña puede incrementar su velocidad lineal
if( (abs(self.robot_vel.Turn)) < 0.1 ):
    #El UMA se queda a 50cm de distancia de separacion del goal y no se mueva si el objetivo esta a mas
    if(dist_to_goal  > (latlong/2.0) and dist_to_goal  < (latlong*500)):
        self.robot_vel.Velocity = self.robot_vel.Velocity + 0.3
        if (self.robot_vel.Velocity >= (kdist*dist_to_goal) ):
            self.robot_vel.Velocity = kdist*dist_to_goal
        if (self.robot_vel.Velocity >= vmax):
            self.robot_vel.Velocity = vmax
    else:
        if ( (abs(self.robot_vel.Turn)) != 0.0 ):
            self.robot_vel.Velocity = 0.001 #Al publicar a cmd ocupa que velocity sea mayor a 0 para pod
        else:
            self.robot_vel.Velocity = 0.0
```

# RESULTS

Fig. 5 UMA in a simulated room with objects going to the goal coordinate

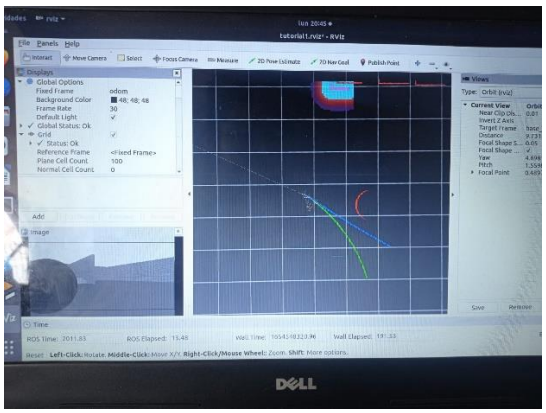


Fig. 6 UMA in a simulated room with objects going to the goal coordinate vision of the robot and rectification of the route to avoid objects.

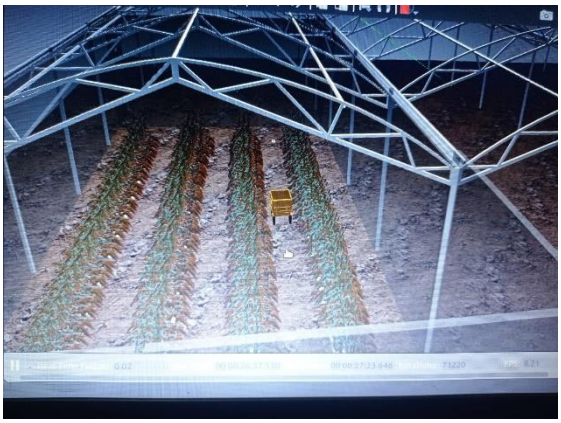


Fig. 7 UMA in a simulated greenhouse going to the goal coordinate.

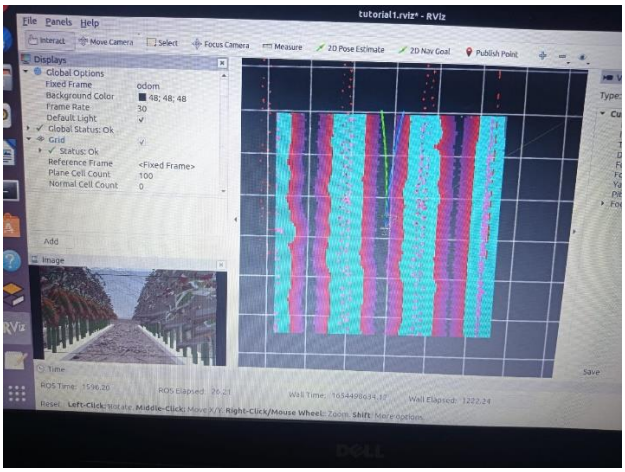


Fig. 8 UMA in a simulated greenhouse going to the goal coordinate avoiding obstacles.

As it can bee seen in the figures 5 to 8, the robot performed as desired, it reaches the goal requested by itself. It is important to mention that it performs better in the room environment because there is more space to make its turns and the reduced space in the greenhouse is an important aspect to take into account because sometimes the robot is not able, for future implementations it will

be crucial to implement a different mechanism of wheels so the robot can rotate in its own axis with out the need of a large space.

works and also to see how to take into account the existing sensors and actuators available today to do an implementation. I think in future implementations I need to upgrade the level of intelligence of my agent to make the harvesting autonomously and also to learn from past.

## CONCLUSIONS

This project was interesting and enriching to understand how an agent

## REFERENCES

[1] Bansall. (2022). Agents in Artificial Intelligence. *GeeksforGeeks*. Recovered from: https://www.geeksforgeeks.org/agents-artificial-intelligence/

[2] Robotisim. (n.d.). Robotisim. Recovered from: https://th.bing.com/th/id/OIP.m6w2Bm1ZWe-eKG16V7B6tgHaDJ?pid=ImgDet&rs=1

[3] VECTORNAV. (n.d.). WHAT IS AN INERTIAL MEASUREMENT UNIT?. *VECTORNAV*. Recovered from: https://www.vectornav.com/resources/inertial-navigation-articles/what-is-an-inertial-measurement-unit-imu?gclid=CjwKCAjwy_aUBhACEiwA2IHHQHDP4UBqZ3jTAqTLxIWA8LOJdSqLFTP-CP0BHIGevbDOQWSGEnsPWBoCpP4QAvD_BwE

[4] NATE. (n.d.). What is GPS RTK?. *Sparkfun-start something*. Recovered from: https://learn.sparkfun.com/tutorials/what-is-gps-rtk/all

[5] ROS. (2021). ROS-Robot Operating System. Recovered from: https://www.ros.org/

[6] B. Arad, J. Balendonck, R. Barth, O. Ben-Shahar, Y. Edan, et al. (2020). Development of a sweet pepper harvesting robot. *Wiley Online Library*. Recovered from: https://onlinelibrary.wiley.com/doi/10.1002/rob.21937

# REFERENCES OF THE IMAGES

[1] Robotisim. (n.d.) Visualising the LiDAR with RVIZ and gazebo. Recovered from: https://www.bing.com/images/search?view=detailV2&ccid=m6w2Bm1Z&id=911E7E54B4FBE3DE0A11E162A9B5F764EC587A9C&thid=OIP.m6w2Bm1ZWe-eKG16V7B6tgHaDJ&mediaurl=https%3a%2f%2frobotisim.com%2fwp-content%2fuploads%2f2021%2f06%2fimage-102-768x326.png&cdnurl=https%3a%2f%2fth.bing.com%2fth%2fid%2fR.9bac36066d5959ef9e286d7a57b07ab6%3frik%3dnHpY7GT3tali4Q%26pid%3dImgRaw%26r%3d0&exph=326&expw=768&q=lidar+in+a+robot+rviz&simid=608011230223748223&FORM=IRPRST&ck=E66FB1D91D90DB985200D81CB072E187&selectedIndex=70&ajaxhist=0&ajaxserp=0

# APPENDIX A

```
R1 = 0;

r2 = 10;

r3 = 15;

theta2_1 = deg2rad(45);

theta2_2 = deg2rad(27.62);

theta2_3 = deg2rad(90);


% Calculations for 45°

theta3_1 = asin((r1-r2*sin(theta2_1))/r3);

r4_1 = r2*cos(theta2_1) + r3*cos(theta3_1);


% Calculations for 27.62°

theta3_2 = asin((r1-r2*sin(theta2_2))/r3);

r4_2 = r2*cos(theta2_2) + r3*cos(theta3_2);


% Calculations for 90°

theta3_3 = asin((r1-r2*sin(theta2_3))/r3);

r4_3 = r2*cos(theta2_3) + r3*cos(theta3_3);


T =
table([rad2deg(theta3_1);r4_1],[rad2deg(theta3_2);r4_2],[rad2deg(theta3_3);
r4_3],'VariableNames',{'At 45°','At 27.62°','At 90°'},
'RowName',{'Theta3','r4'});

disp(T)
```