Lab report for CISC 3060 Lab Assignment #3

Fall 2022

Jan C. Bierowiec

Isabella Rocconova, Margot Dugan, Aaron Leder

1.0 Objective

The objective of lab assignment #3, was to experiment using a TurtleBot3, how to control the robot through manual commands, and how to manipulate certain pieces of code enabling the robot to use laser sensing in order to track the boundaries of the “boxed world” it was located in. TurtleBot is a ROS standard platform robot, which is popular in use among developers and students. There are three versions of the TurtleBot model, the one being used during the lab experiment being TurtleBot3. TurtleBot3 was developed with features to supplement the lacking functions of its predecessors, TurtleBot1 and TurtleBot2, and the demands of users. The TurtleBot3 adopts ROBOTIS smart actuator DYNAMIXEL for driving. TurtleBot3 is useful to understand and test ideas in robotics because TurtleBot3 robots are small, affordable, and programmable, which is ideal for uses in education, research, hobby, and product prototyping. In the lab when changing the code, so that the robot can sense boundaries and move from one place to another, it was fairly easy for the robot to make 360 degree rotations and have a decently accurate distance sensor. The TurtleBot3 used in the experiment was of the Burger model, which is the smallest of the TurtleBot3 family. The TurtleBot3’s core technology is SLAM, Navigation, and Manipulation, making it suitable for home service robots. The TurtleBot can run SLAM(simultaneous localization and mapping) algorithms to build a map and can drive around your room. Also, it can be controlled remotely from a laptop,which was utilized during the lab experiment.

2.0 TurtleBot3 Enclosed Environment Experiment

2.1 Step 1

For Step 1 of the lab, the group had to bring up the TurtleBot3. Each group was paired with one robot and one laptop. When the robot would be turned on, it would be placed in a square enclosure which was the world it would move around in. Once placed in the enclosed square world, and the laser ranger on the top of the robot would start spinning, then the robot was ready to communicate with the laptop. Once logged into the laptop, five terminals were

brought up to run the robot, or rather communicate with it. All terminals were in the CISC3060 directory. The first window roscore -p11315 was typed in, with 5 being the port number written on the T3. Once this started to run, starting to spin the laser ranger, this window was minimized, and another terminal was opened to ssh into the T3, typing the command, ssh pi@10.10.3.163, with 163 being the number on the robot. Once this command was written a password was prompted to be inputted, that being turtlebot. Once the T3 was connected with the laptop, roslaunch turtlebot3_initialize-t.launch was typed in the terminal which resulted in an output of messages, which terminated with the message “Calibration End”. Once this was completed, another terminal was brought up, with the command roslaunch tutlebot3_initialize-lnr.luanch, which would run smaller diagnostics on the robot. Once this was completed, another terminal window was opened and rostopic list was typed into the terminal. This provided a list of active ROS topics which included /odom, /scan, and /cmd_vel. In Figure 1, below a screenshot of the terminals is shown below, and in Figure 2, there is an image of the robot, in the square enclosed world.

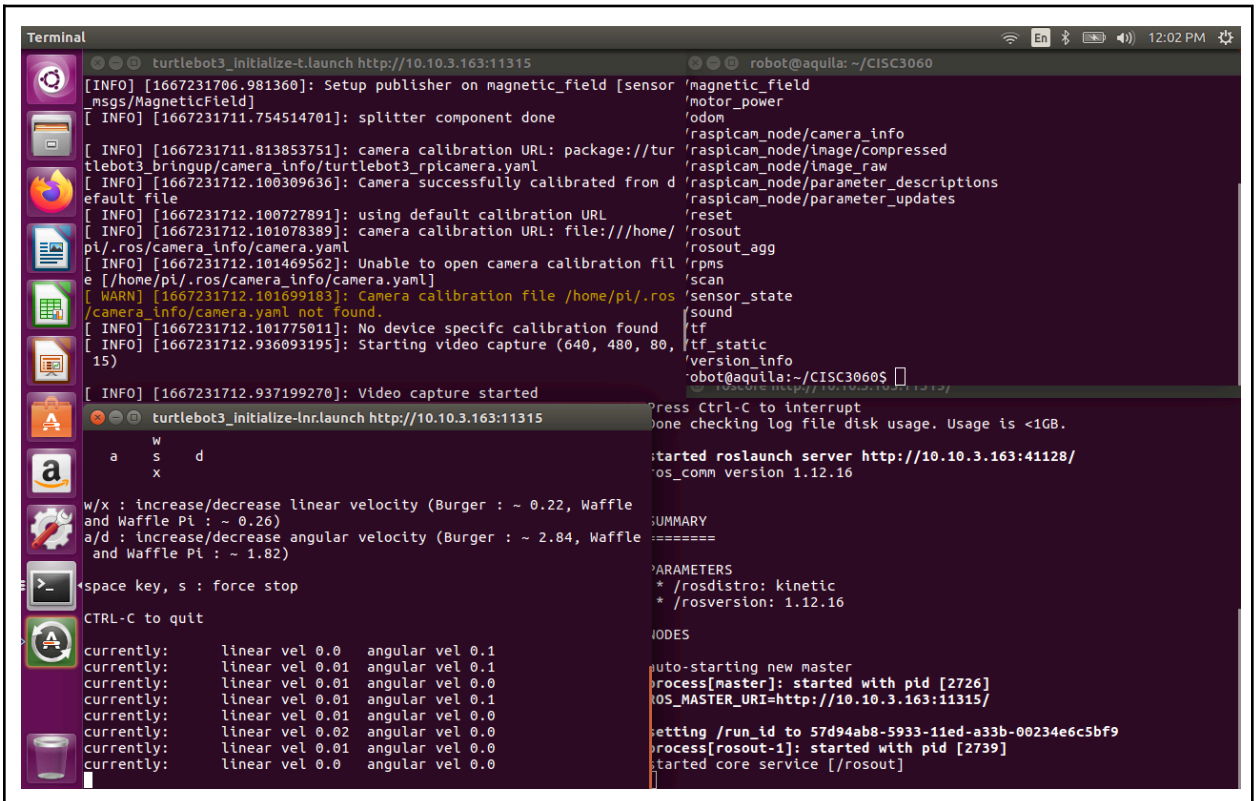


Figure 1: A screenshot of the terminals brought up on the laptop screen to communicate with the robot. Bottom Right terminal is Terminal 1. Upper Left terminal is Terminal 2. Bottom Left terminal is Terminal 3. Upper Right terminal is Terminal 4.

Figure 2: Here is a screenshot of the TurtleBot3 placed in the square-like environment ready to be tested.

2.2 Step 2

For Step 2 of the lab RViz was brought up, which was used to see the laser range points around the robot based on where the robot was placed. Since RViz was not running for the group originally, a command rosrun rviz rviz -d `rospack find turtlebot3_description`/rviz/model.rviz was run in order to see the virtual world the robot was in on the computer in relation to the actual square environment it was in. Once RViz was running, a group member would enter the square environment the robot was in, and would manually move the robot to each of the four corners of the square environment. Doing so provided different outputs of the laser boundary points the robot sensed. Figures 3 through 6 demonstrate the robot and the laser points along the square world boundaries when the robot was placed at the corners of the square world. Figure 7 demonstrates the robot and laser points along the square world boundaries when the robot was placed in the middle of the square world. Using RViz helped record and visualize the sensing of the robot and the environment around it. A further analysis of these images will be provided in Section 3.0.

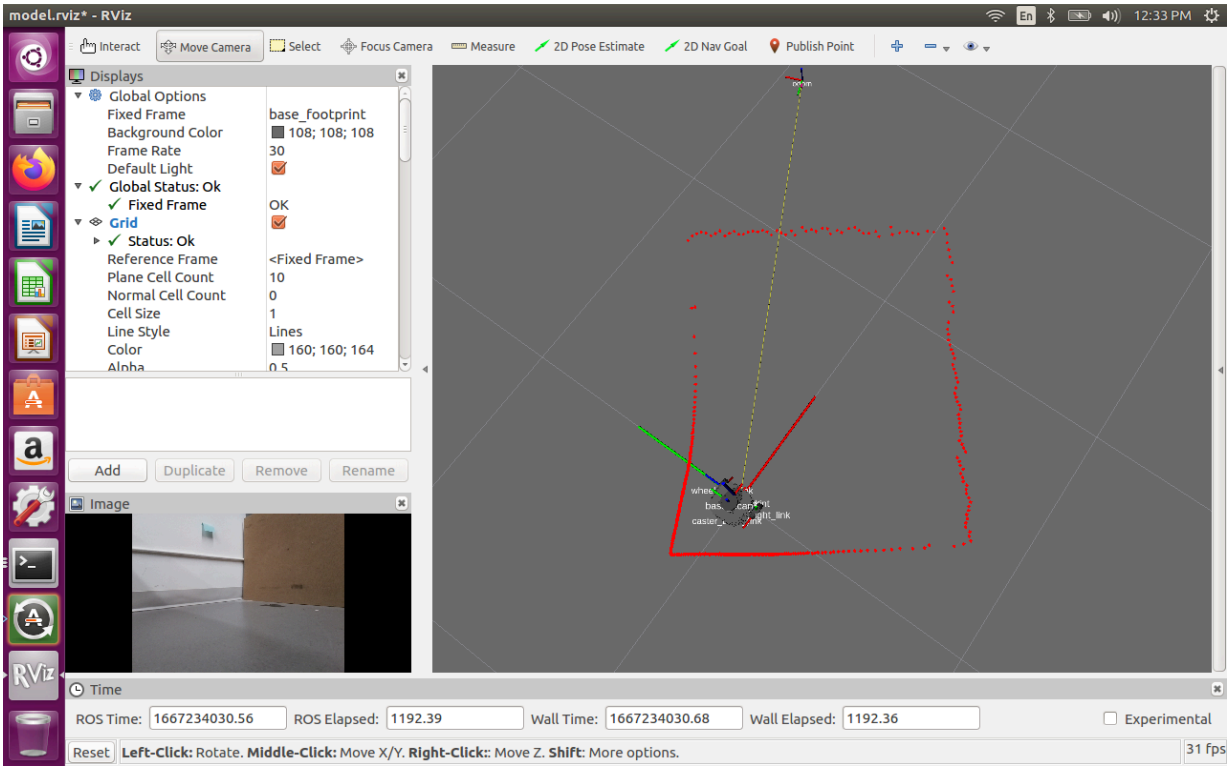


Figure 3: The T3 is placed at the southwest corner of the square-like enclosed world. The T3 reads in the laser points of the environment around it, with the strongest points being those directly next to and behind the robot, gaps in points being the northwest and southeast corners, and ragged but somewhat linear points along the north and east sides. The camera of the robot is looking at the northeast corner of the environment.

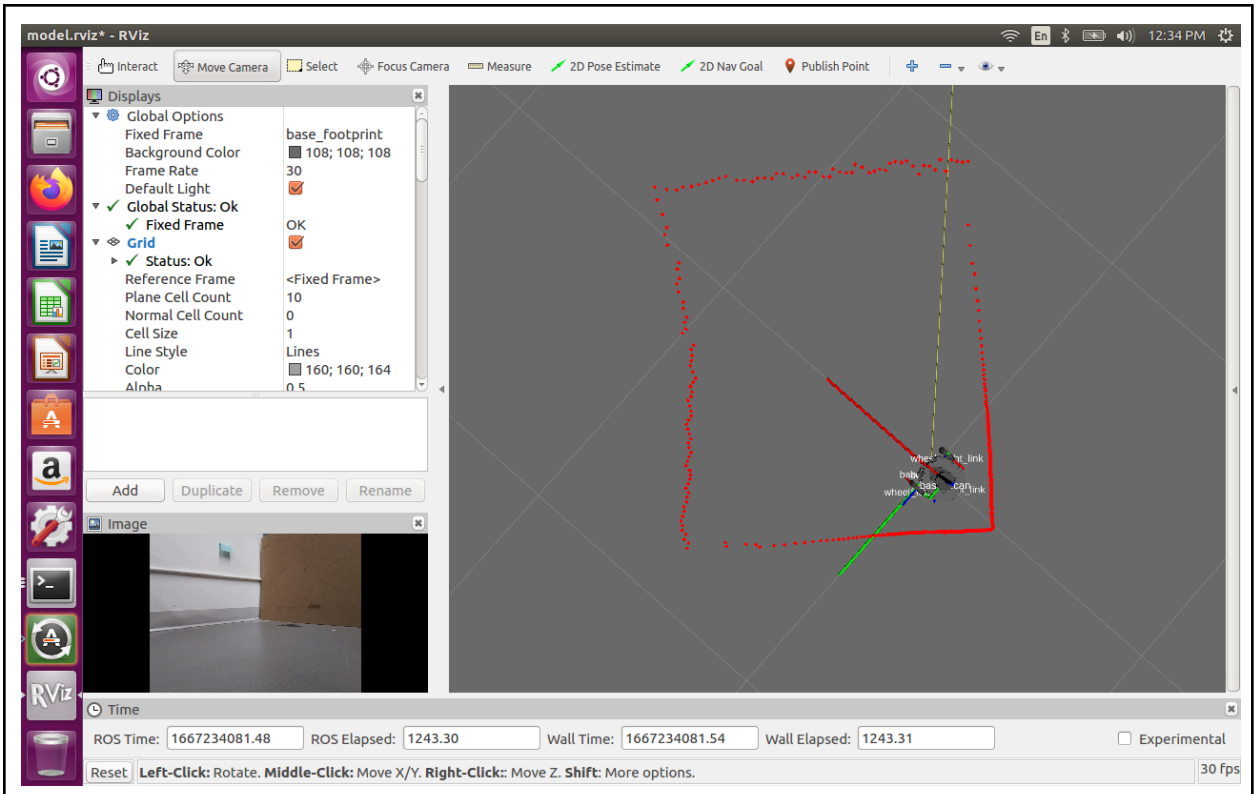


Figure 4: The T3 is placed at the southeast corner of the square-like enclosed world. The T3 reads in the laser points of the environment around it, with the strongest points being those directly next to and behind the robot, gaps in points being the northeast and southwest corners, and ragged but somewhat linear points along the north and west sides. The camera of the robot is looking at the northeast corner of the environment, because the change in position has not been registered yet.

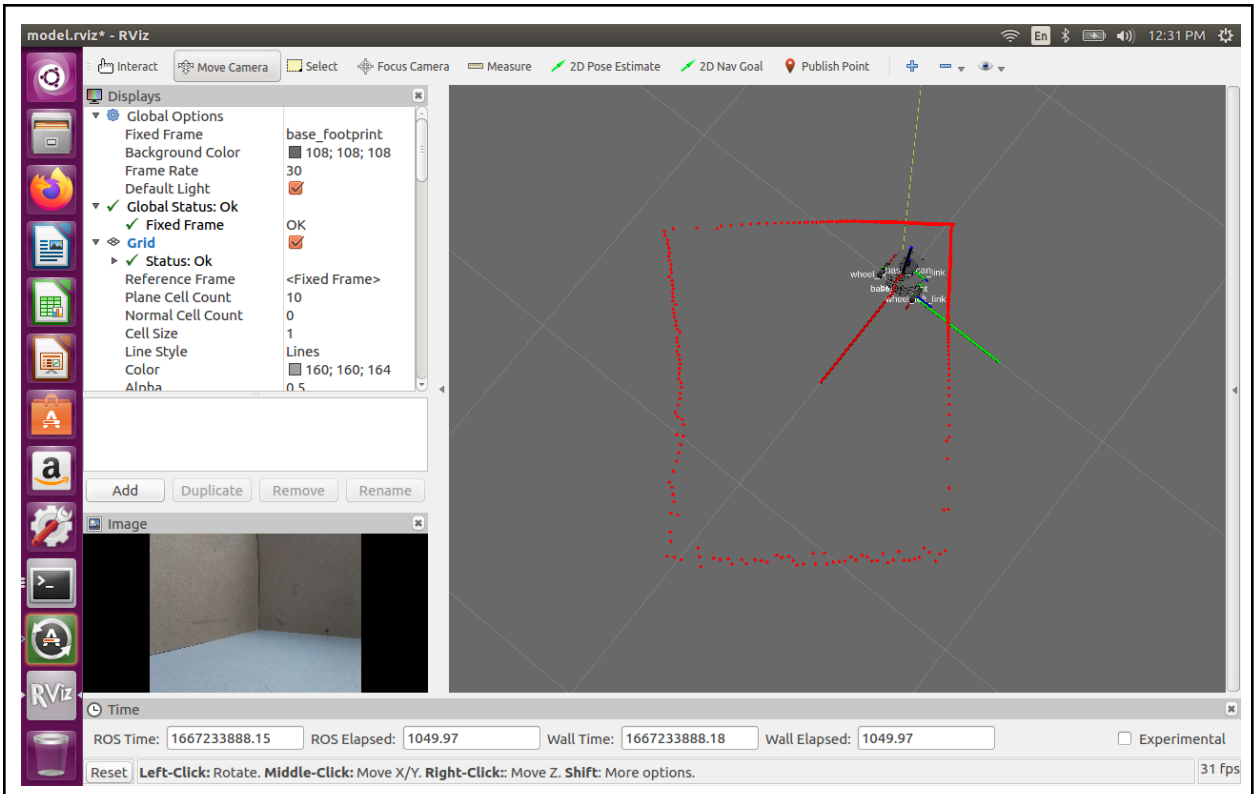


Figure 5: The T3 is placed at the northeast corner of the square-like enclosed world. The T3 reads in the laser points of the environment around it, with the strongest points being those directly next to and behind the robot, gaps in points being the northeast and southeast corners, and ragged but somewhat linear points along the south and west sides. The camera of the robot is looking at the southeast corner of the environment.

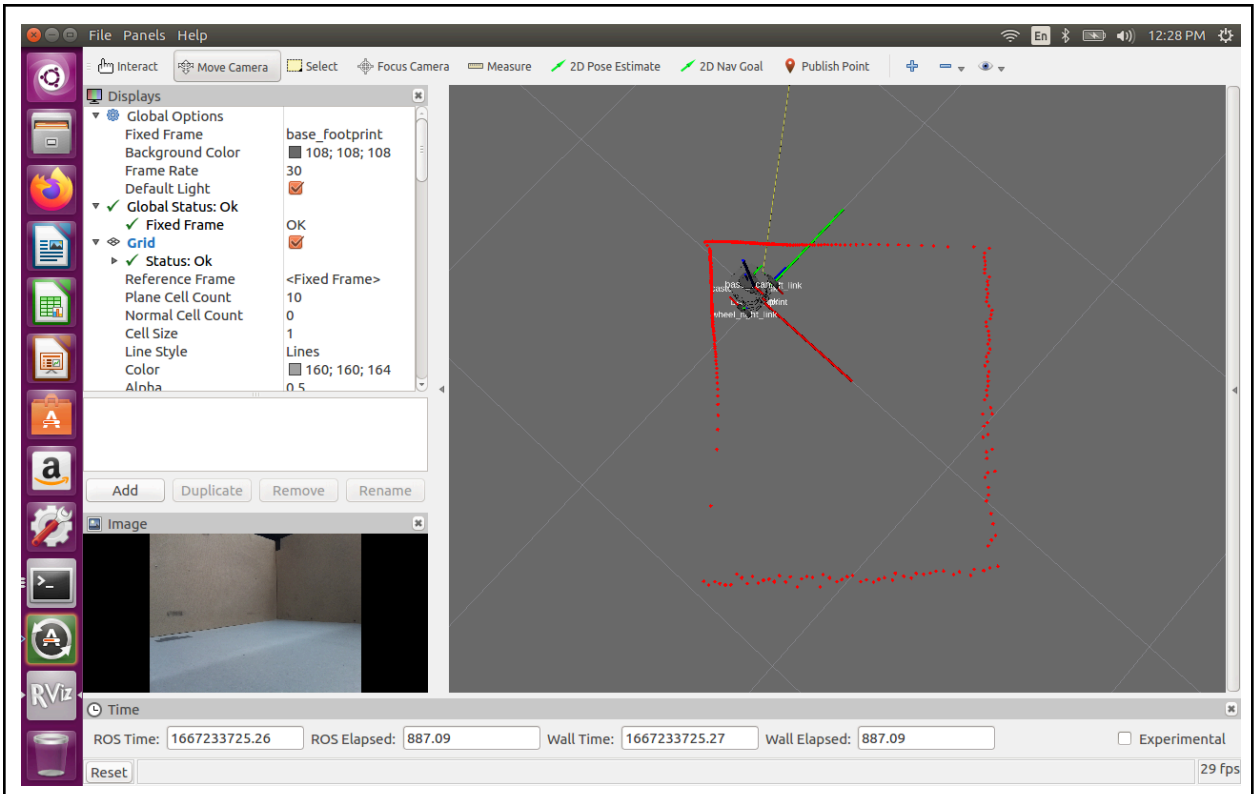


Figure 6: The T3 is placed at the northwest corner of the square-like enclosed world. The T3 reads in the laser points of the environment around it, with the strongest points being those directly next to and behind the robot, gaps in points being the northeast and southwest corners, and ragged but somewhat linear points along the south and east sides. The camera of the robot is looking at the southeast corner of the environment.

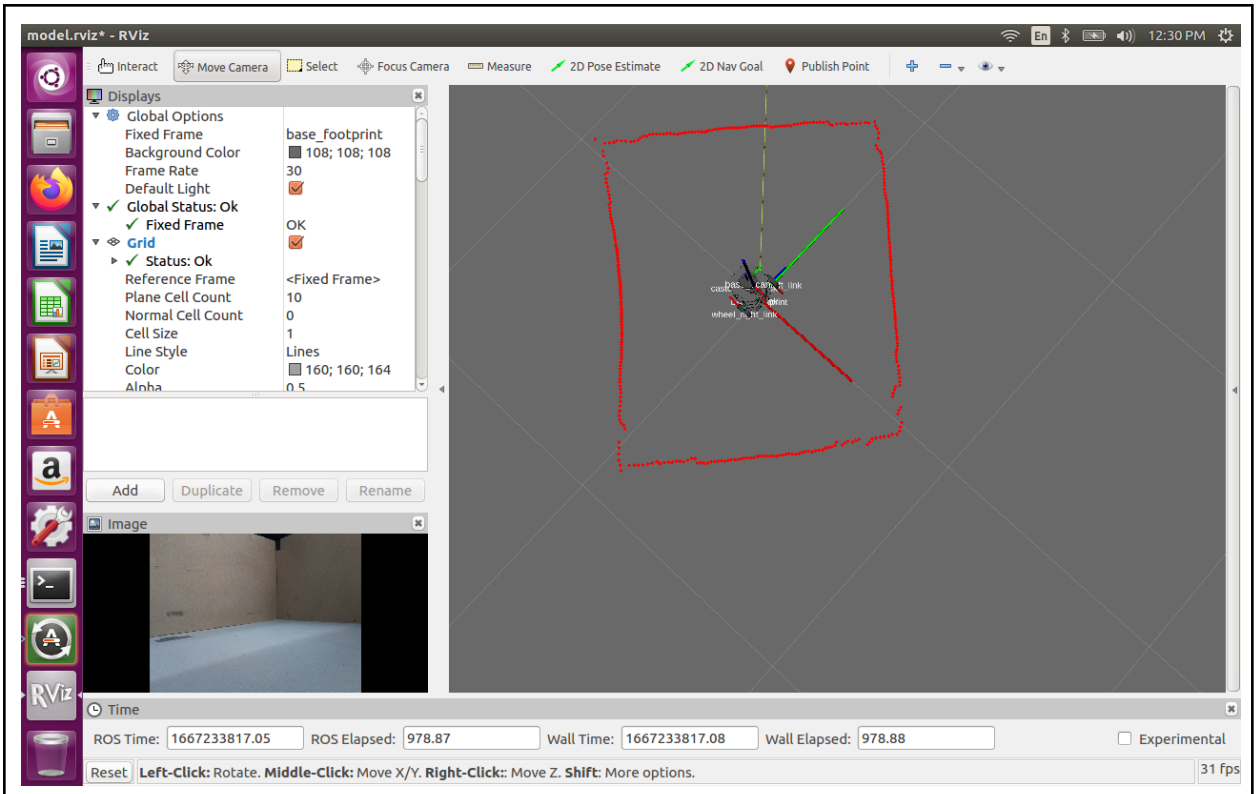


Figure 7: The T3 is placed at the center of the square-like enclosed world. The T3 reads in the laser points of the environment around it, with the laser points around it being somewhat linear around it. There are no major gaps in points like when the robot was placed in different corners. When placed at the center, the robot reads in mostly the whole environment equally. The camera of the robot is looking at the southeast corner of the environment, since it is facing that direction.

2.3 Step 3

For step 3 of the lab a copy of the Wander.py program was downloaded from BlackBoard, and renamed wander.py which was then stored in the CISC3060 folder on the laptop. This file was then modified and run which communicated with the robot. Most of the code in the wander.py program did not need to be changed, only the ranges for the distances and velocities were changed. The original wander.py program can be seen below in Figure 8.

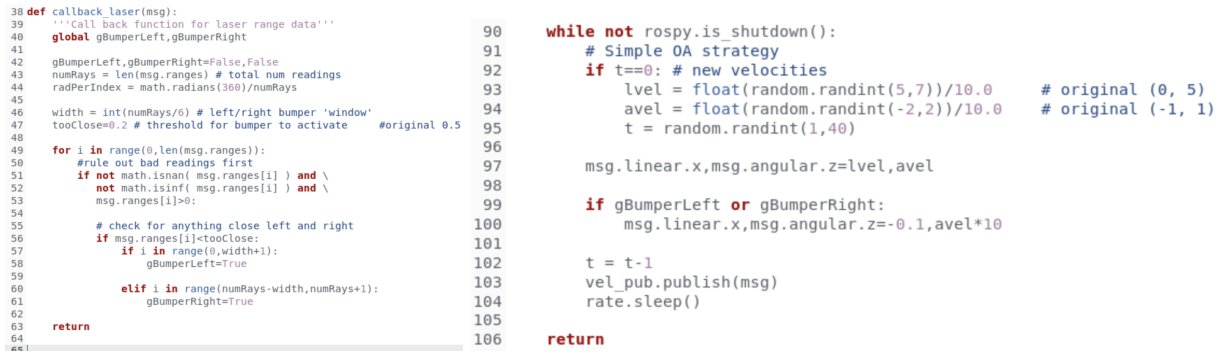
```
#!/usr/bin/env python
# license removed for brevity
#
# Example program that
# wanders,
# dml 2020
#
#

import math
import random

import rospy # needed for ROS
import numpy as np # for map arrays
import matplotlib.pyplot as plt

from geometry_msgs.msg import Twist      # ROS Twist message
from sensor_msgs.msg import LaserScan    # ROS laser msg




# ROS Topics
#
motionTopic='/cmd_vel' # turtlebot vel topic
laserTopic = '/scan'   # laser ranging topic

#global variable, to communicate with callbacks

gBumperLeft,gBumperRight= False, False # left/right close




# callback for the laser range data
# just sets the global Bump flags
#
def callback_laser(msg):
    '''Call back function for laser range data'''
    global gBumperLeft,gBumperRight

    gBumperLeft,gBumperRight=False,False
    numRays = len(msg.ranges) # total num readings
    radPerIndex = math.radians(360)/numRays

    width = int(numRays/6) # left/right bumper 'window'
    tooClose=0.5 # threshold for bumper to activate

    for i in range(0,len(msg.ranges)):
        #rule out bad readings first
        if not math.isnan( msg.ranges[i] ) and \
           not math.isinf( msg.ranges[i] ) and \
           msg.ranges[i]>0:

            # check for anything close left and right
            if msg.ranges[i]<tooClose:
                if i in range(0,width+1):
                    gBumperLeft=True

                elif i in range(numRays-width,numRays+1):
                    gBumperRight=True

    return


# wander_node -
# Moves frandomly until it detects a close surface
# then avoids it

def wander_node():
    '''continually move forward until a close surface is detected'''
    global gBumper

    # all ROS 'nodes' (ie your program) have to do the following
    rospy.init_node('Wander', anonymous=True)
```

```
def wander_node():
    '''continually move forward until a close surface is detected'''
    global gBumper

    # all ROS 'nodes' (ie your program) have to do the following
    rospy.init_node('Wander', anonymous=True)

    # register as a ROS publisher for the velocity topic
    vel_pub = rospy.Publisher(motionTopic, Twist, queue_size=10)
    # register as a subscribe for the laser scan topic
    scan_sub = rospy.Subscriber(laserTopic, LaserScan, callback_laser)

    # this is how frequently the loop below iterates
    rate = rospy.Rate(10) # Hz

    msg = Twist() # new velocity message
    msg.linear.x,msg.angular.z=0,0
    vel_pub.publish(msg) # stop all motors
    t = 0

    while not rospy.is_shutdown():
        # Simple OA strategy
        if t==0: # new velocities
            lvel = float(random.randint(0,5))/10.0
            avel = float(random.randint(-1,1))/10.0
            t = random.randint(1,40)

        msg.linear.x,msg.angular.z=lvel,avel

        if gBumperLeft or gBumperRight:
            msg.linear.x,msg.angular.z=-0.1,avel*10

        t = t-1
        vel_pub.publish(msg)
        rate.sleep()

    return

#
# This function is called by ROS when you stop ROS
# Here we use it to send a zero velocity to robot
# in case it was moving when you stopped ROS
#

def callback_shutdown():
    print("Shutting down")
    pub = rospy.Publisher(motionTopic, Twist, queue_size=10)
    msg = Twist()
    msg.angular.z=0.0
    msg.linear.x=0.0
    pub.publish(msg)
    return



#------------------------------MAIN  program---------------------
if __name__ == '__main__':
    try:
        rospy.on_shutdown(callback_shutdown)
        wander_node()
    except rospy.ROSInterruptException:
        pass
```

Figure 8: Here is the original wander.py program before adjustments were made to the program.

In the laser callback function line 46 controls the number of laser rays in front of the robot that are checked for proximity, (numRays/6) and line 47 the proximity of distance 0.5m. In the wander_node function, line 93 which had the linear velocity (lvel) ranged from 0 to 5, and line 94 which had the angular velocity (avel) ranged from -1 to 1. For the first test, the values were changed from 0.5 to 0.2 for the proximity distance since the square-like environment around the robot was relatively small, and the linear velocity was changed to a range from 5 to 7. Then the angular velocity was changed to range from -2 to 2. With these coordinates, the robot at first did not move for a minute, then when the robot started to move, it would hit the

boundaries around it and the robot would struggle to move around the boundary finding empty space. The change in code can be seen below in Figure 9.

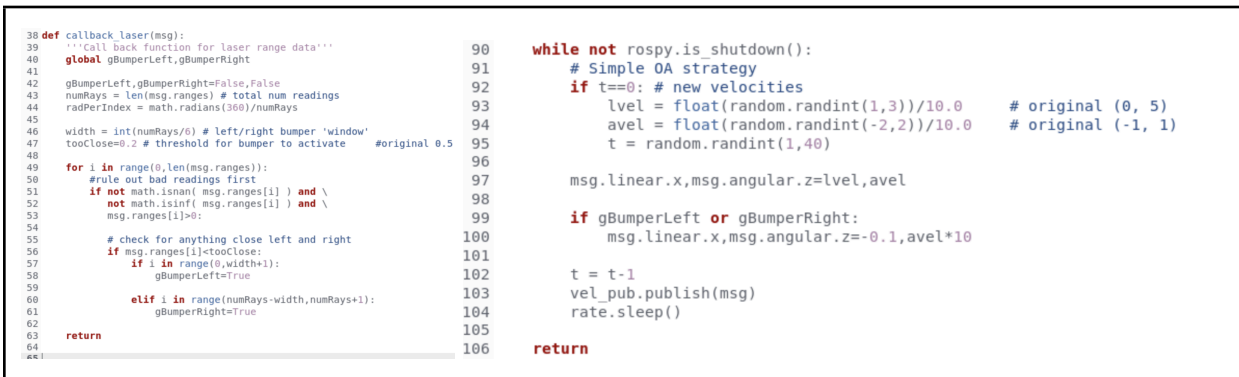


```
def callback_laser(msg):
    '''Call back function for laser range data'''
    global gBumperLeft,gBumperRight

    gBumperLeft,gBumperRight=False,False
    numRays = len(msg.ranges) # total num readings
    radPerIndex = math.radians(360)/numRays

    width = int(numRays/6) # left/right bumper 'window'
    tooClose=0.2 # threshold for bumper to activate     #original 0.5

    for i in range(0,len(msg.ranges)):
        #rule out bad readings first
        if not math.isnan( msg.ranges[i] ) and \
           not math.isinf( msg.ranges[i] ) and \
           msg.ranges[i]>0:

            # check for anything close left and right
            if msg.ranges[i]<tooClose:
                if i in range(0,width+1):
                    gBumperLeft=True

                elif i in range(numRays-width,numRays+1):
                    gBumperRight=True

    return
```

```
    while not rospy.is_shutdown():
        # Simple OA strategy
        if t==0: # new velocities
            lvel = float(random.randint(5,7))/10.0     # original (0, 5)
            avel = float(random.randint(-2,2))/10.0    # original (-1, 1)
            t = random.randint(1,40)

        msg.linear.x,msg.angular.z=lvel,avel

        if gBumperLeft or gBumperRight:
            msg.linear.x,msg.angular.z=-0.1,avel*10

        t = t-1
        vel_pub.publish(msg)
        rate.sleep()

    return
```

Figure 9: Here is the adjusted code for the first test run. The proximity of distance is set to be 0.2, the lvel ranges from 5 to 7, and the avel ranges from -2 to 2.

Observing this behavior, another test was conducted. Since the robot was moving too fast and it took a decent amount of time to turn around after it hit the wall, it was deduced that the linear velocity should be changed. The proximity of distance was still kept to be 0.2m, and the angular velocity ranged from -2 to 2, however the linear velocity was changed from a range of 5 to 7 to 1 and 3. This caused the robot to not hit the boundaries of the environment it was in too frequently, which was good, but the robot would move in increments without directly traveling and stopping at a boundary. With these coordinates inputted the robot would move in empty space, stop then move again. Although these coordinates made the robot slower, the robot was able to sense the environment boundaries more accurately without hitting the wall for a longer period of time. The change in code can be seen below in Figure 10.



```
def callback_laser(msg):
    '''Call back function for laser range data'''
    global gBumperLeft,gBumperRight

    gBumperLeft,gBumperRight=False,False
    numRays = len(msg.ranges) # total num readings
    radPerIndex = math.radians(360)/numRays

    width = int(numRays/6) # left/right bumper 'window'
    tooClose=0.2 # threshold for bumper to activate     #original 0.5

    for i in range(0,len(msg.ranges)):
        #rule out bad readings first
        if not math.isnan( msg.ranges[i] ) and \
           not math.isinf( msg.ranges[i] ) and \
           msg.ranges[i]>0:

            # check for anything close left and right
            if msg.ranges[i]<tooClose:
                if i in range(0,width+1):
                    gBumperLeft=True

                elif i in range(numRays-width,numRays+1):
                    gBumperRight=True

    return
```

```
    while not rospy.is_shutdown():
        # Simple OA strategy
        if t==0: # new velocities
            lvel = float(random.randint(1,3))/10.0     # original (0, 5)
            avel = float(random.randint(-2,2))/10.0    # original (-1, 1)
            t = random.randint(1,40)

        msg.linear.x,msg.angular.z=lvel,avel

        if gBumperLeft or gBumperRight:
            msg.linear.x,msg.angular.z=-0.1,avel*10

        t = t-1
        vel_pub.publish(msg)
        rate.sleep()

    return
```

Figure 10: Here is the adjusted code for the second test run. The proximity of distance is set to be 0.2, the lvel ranges from 1 to 3, and the avel ranges from -2 to 2.

Upon observing this change of behavior and improvement of the robot sensing, another test was performed. This time however, the proximity of distance was changed from 0.2 to 0.3m. This is because the robot started to move slowly and as it neared the boundaries, but still hit them. With a smaller proximity of distance the robot might hit the boundary or not hit it at all. The linear and the angular velocities of the robot were not changed. When the program was run, the robot turned out to not move in small increments like in the second test. The robot was also able to pause and sense accurately the boundaries of the environment it was in, without much of a struggle going back to where there was more space to travel. The change in code can be seen below in Figure 11.

```
    while not rospy.is_shutdown():
        # Simple OA strategy
        if t==0: # new velocities
            lvel = float(random.randint(1,3))/10.0     # original (0, 5)
            avel = float(random.randint(-2,2))/10.0    # original (-1, 1)
            t = random.randint(1,40)

        msg.linear.x,msg.angular.z=lvel,avel

        if gBumperLeft or gBumperRight:
            msg.linear.x,msg.angular.z=-0.1,avel*10

        t = t-1
        vel_pub.publish(msg)
        rate.sleep()

    return
```

```
def callback_laser(msg):
    '''Call back function for laser range data'''
    global gBumperLeft,gBumperRight

    gBumperLeft,gBumperRight=False,False
    numRays = len(msg.ranges) # total num readings
    radPerIndex = math.radians(360)/numRays

    width = int(numRays/6) # left/right bumper 'window'
    tooClose=0.3 # threshold for bumper to activate     #original 0.5

    for i in range(0,len(msg.ranges)):
        #rule out bad readings first
        if not math.isnan( msg.ranges[i] ) and \
           not math.isinf( msg.ranges[i] ) and \
           msg.ranges[i]>0:

           # check for anything close left and right
           if msg.ranges[i]<tooClose:
               if i in range(0,width+1):
                   gBumperLeft=True

               elif i in range(numRays-width,numRays+1):
                   gBumperRight=True

    return
```

Figure 11: Here is the adjusted code for the third test run. The proximity of distance is set to be 0.3, the lvel ranges from 1 to 3, and the avel ranges from -2 to 2.

2.4 Step 4

For the initial test, the group chose the values at random to see how the robot would respond to sensing the boundaries in its environment and how fast it would move. One team member would change the wander.py file, with the different number inputs, and the other members would record the movement of the robot with phones. Once the robot would run for a while, the program would then be killed and the new inputs for the tests mentioned above would be inputted. This was done three times for each test to see how the robot would react to the boundaries of its environment, and the values were changed to see what would be the most optimal way for the robot to sense boundaries without hitting them, or if doing so hitting them gently, and quickly turning to find open space in the square-size world environment. Below in Figures 12 through 15 are the photographs taken of the robot when viewing its movement in RViz. The videos of the robot during the test runs of the linear velocity being (5,7) and (1,3) can be found here respectively, ((5,7) first test), ((1,3) second test), and ((1,3) thirdtest). Lastly Figure 16 shows an image of the world around the robot and how it traveled.

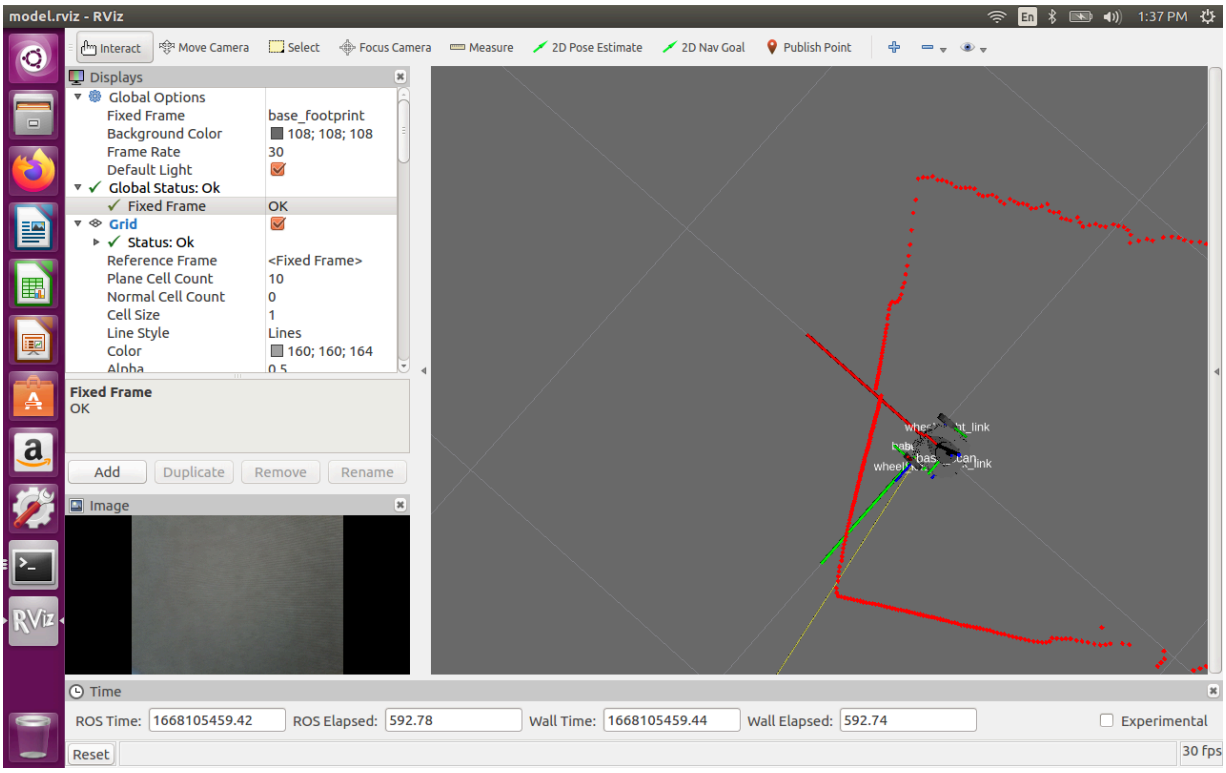


Figure 12: Here is the T3 robot when it approached the cardboard on the northside. The robot is close to the wall and here it was struggling to turn around finding the space to wander.

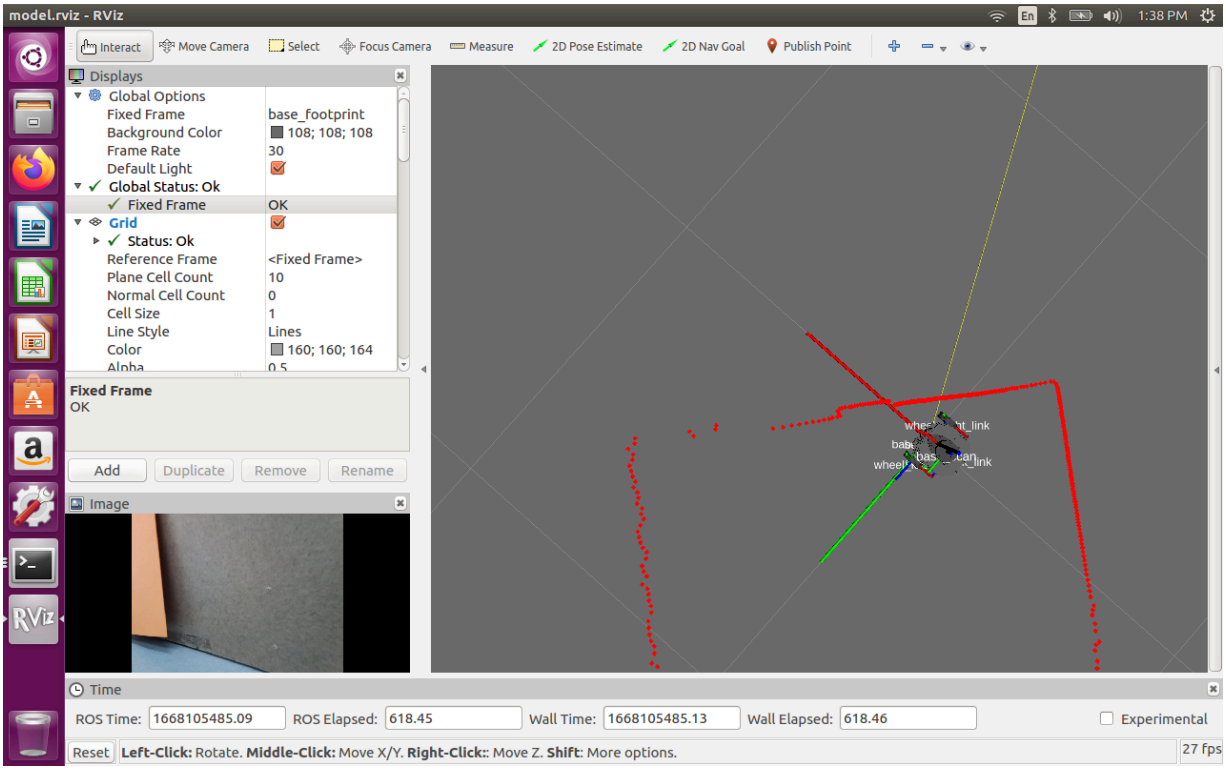


Figure 13: Here the T3 robot is turning around as can be seen both by the image and the camera on the robot. The robot reached a boundary and it is turning around as a result of sensing that boundary looking for wandering space.

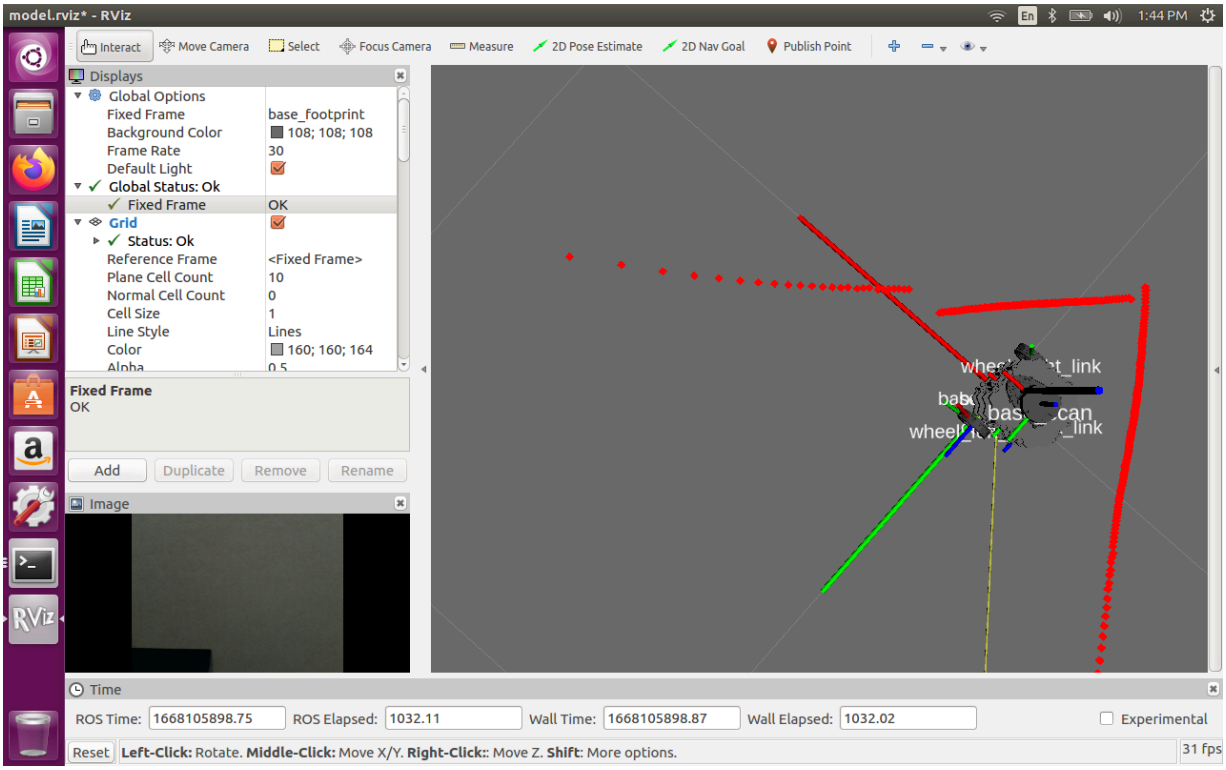


Figure 14: In this image, the t3 robot is at a corner and is rotating about itself turning back from where it came from since at a corner, the robot had a limited area to go to.

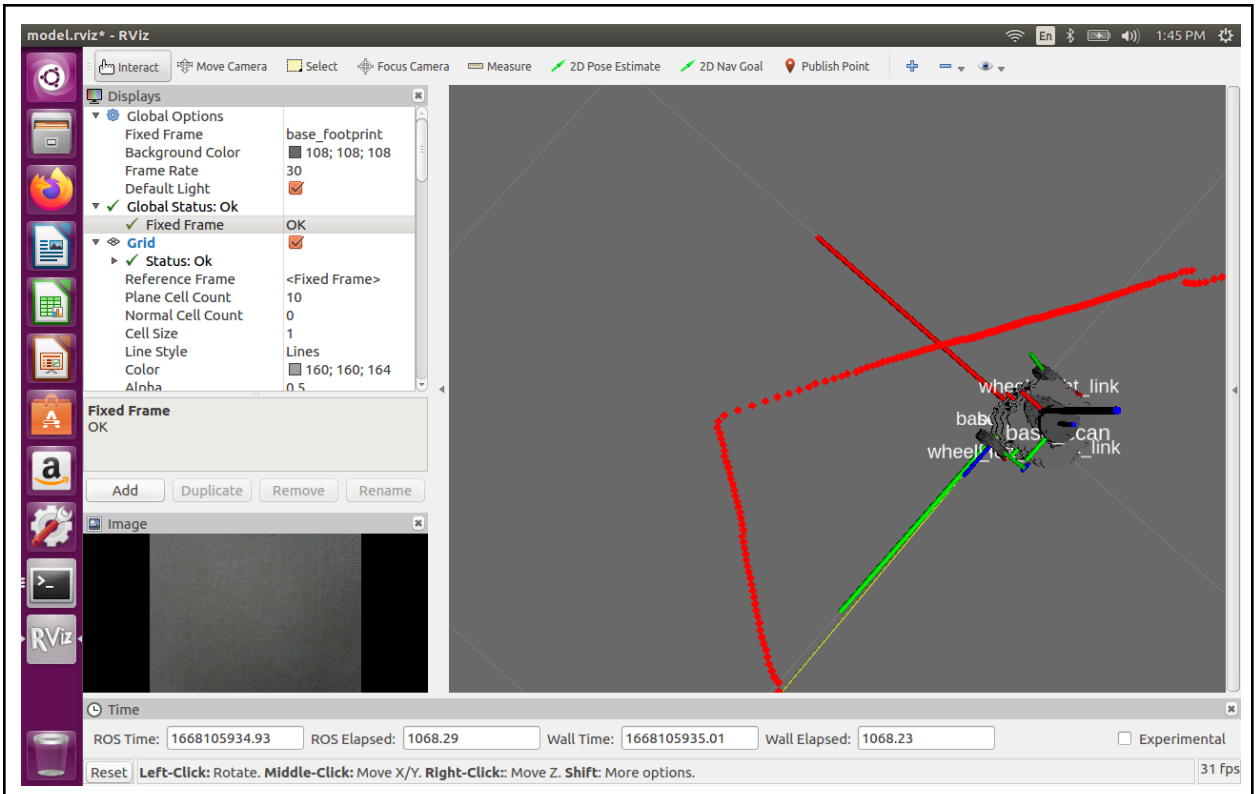


Figure 15: Here is another instance in which the T3 robot reached a boundary, being a cardboard wall, and the robot rotating to the left finding ope space to evade the boundary.

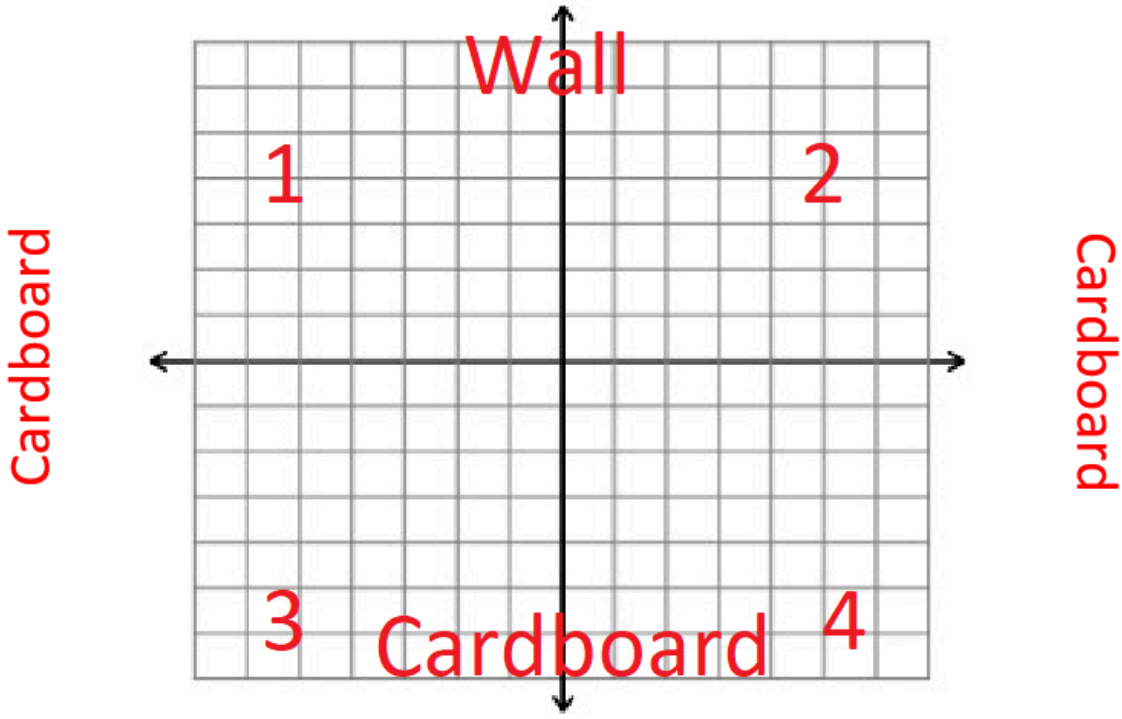


Figure 16: Here is a grid of the square-like enclosed world the T3 robot was in. The robot was surrounded by cardboard on the south, east and west, and surrounded by a wall in the north.

4.0 References

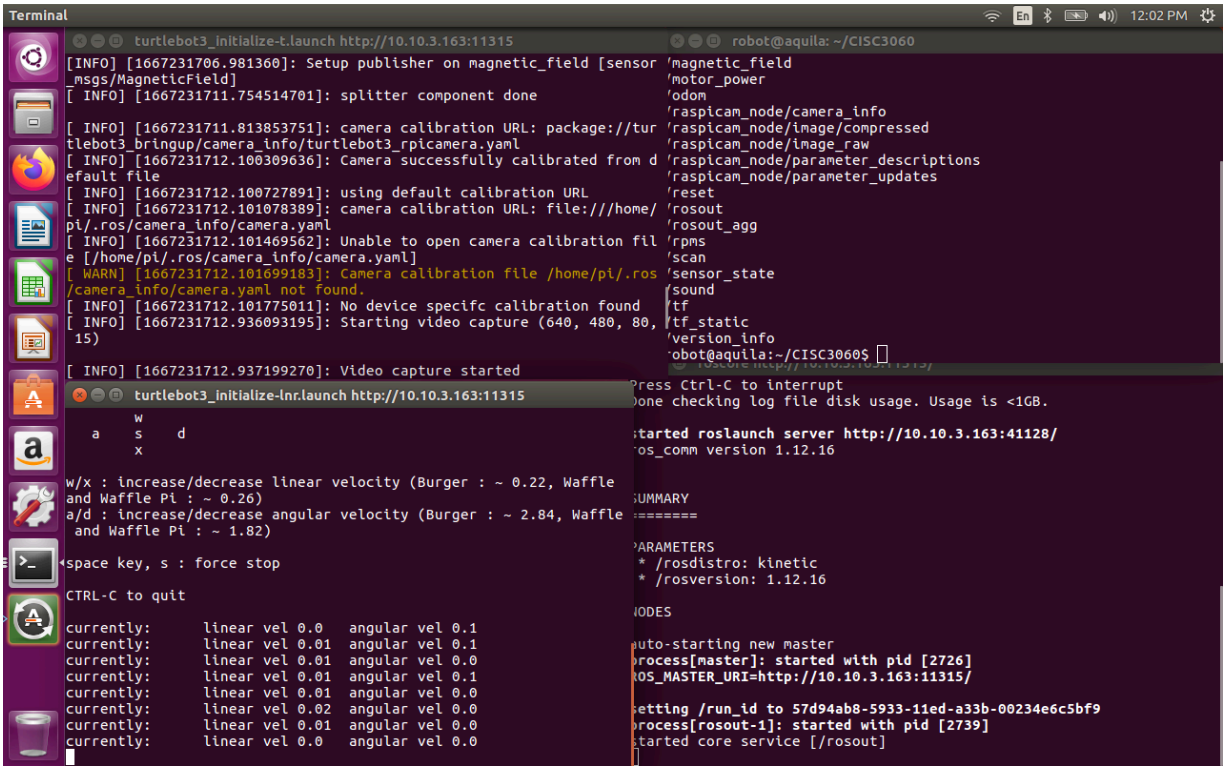


Figure 1: Screenshot of the terminals opened to run the T3 robot.

Figure 2: Screenshot of the T3 robot in its enclosed square world.

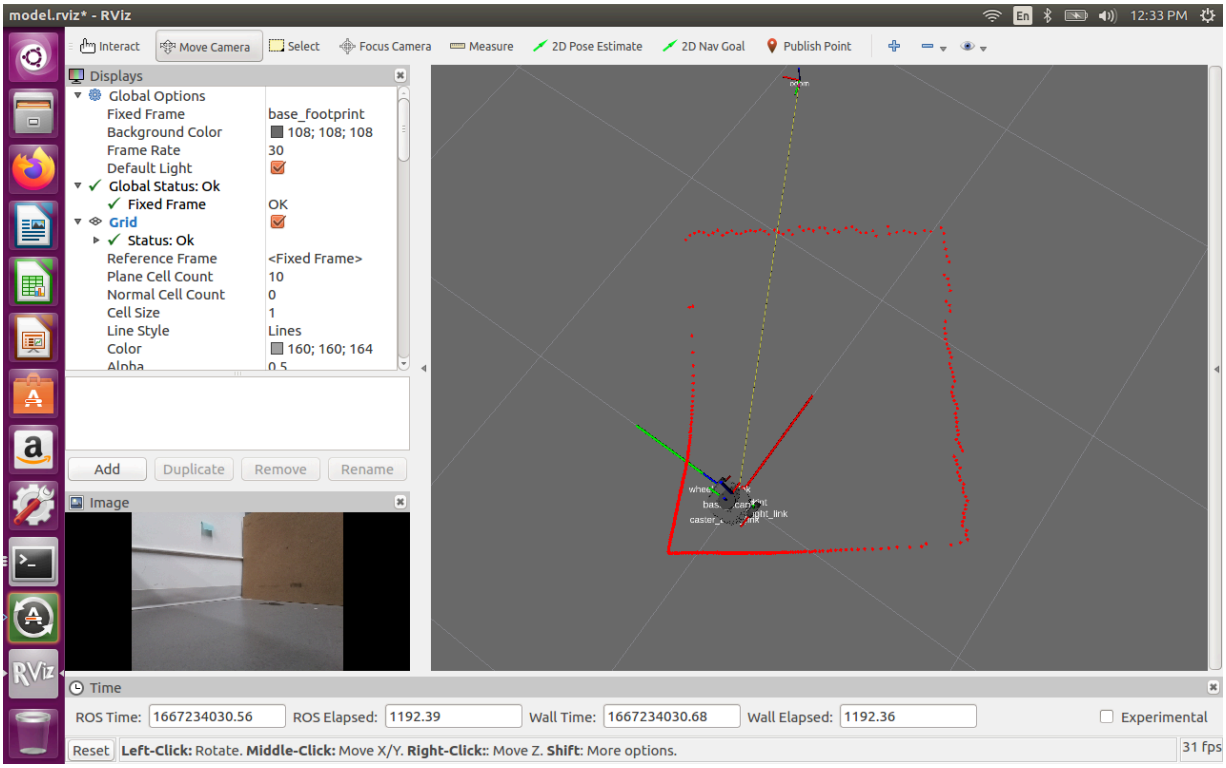


Figure 3: Step 2 T3 Robot placed at the southeast corner.

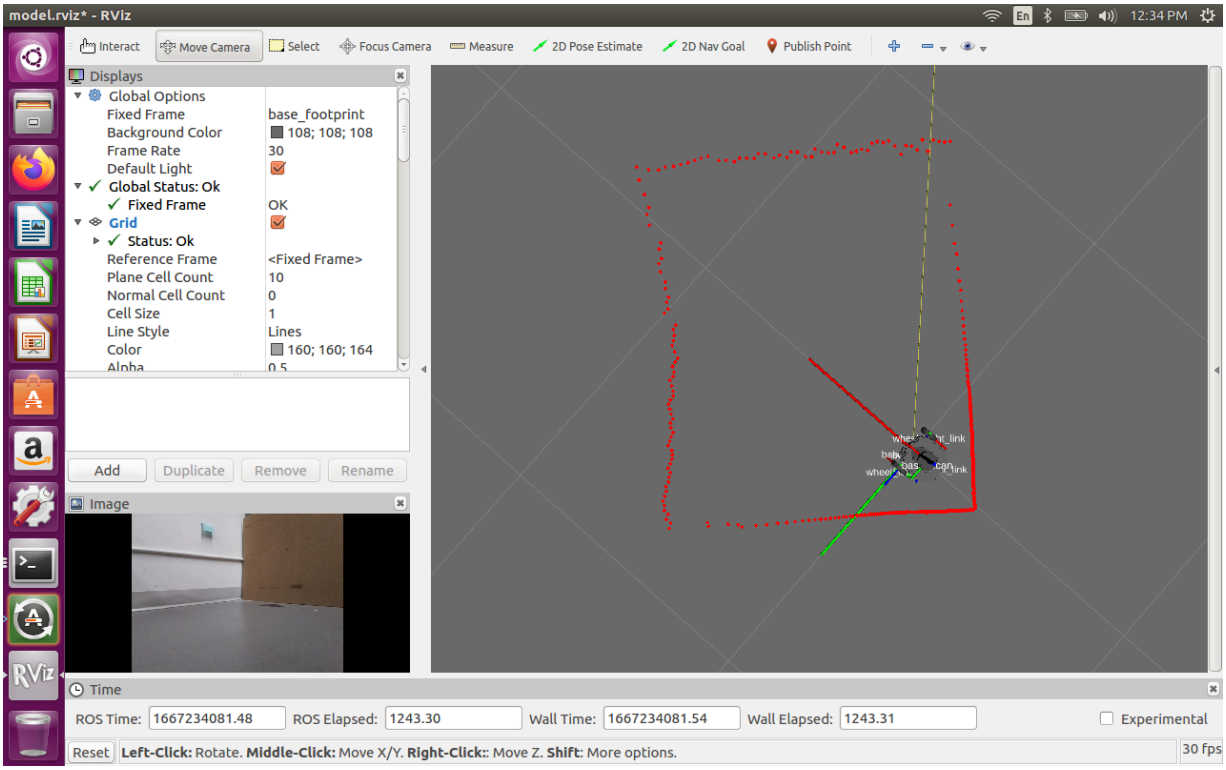


Figure 4: Step 2 T3 Robot placed at the southwest corner.

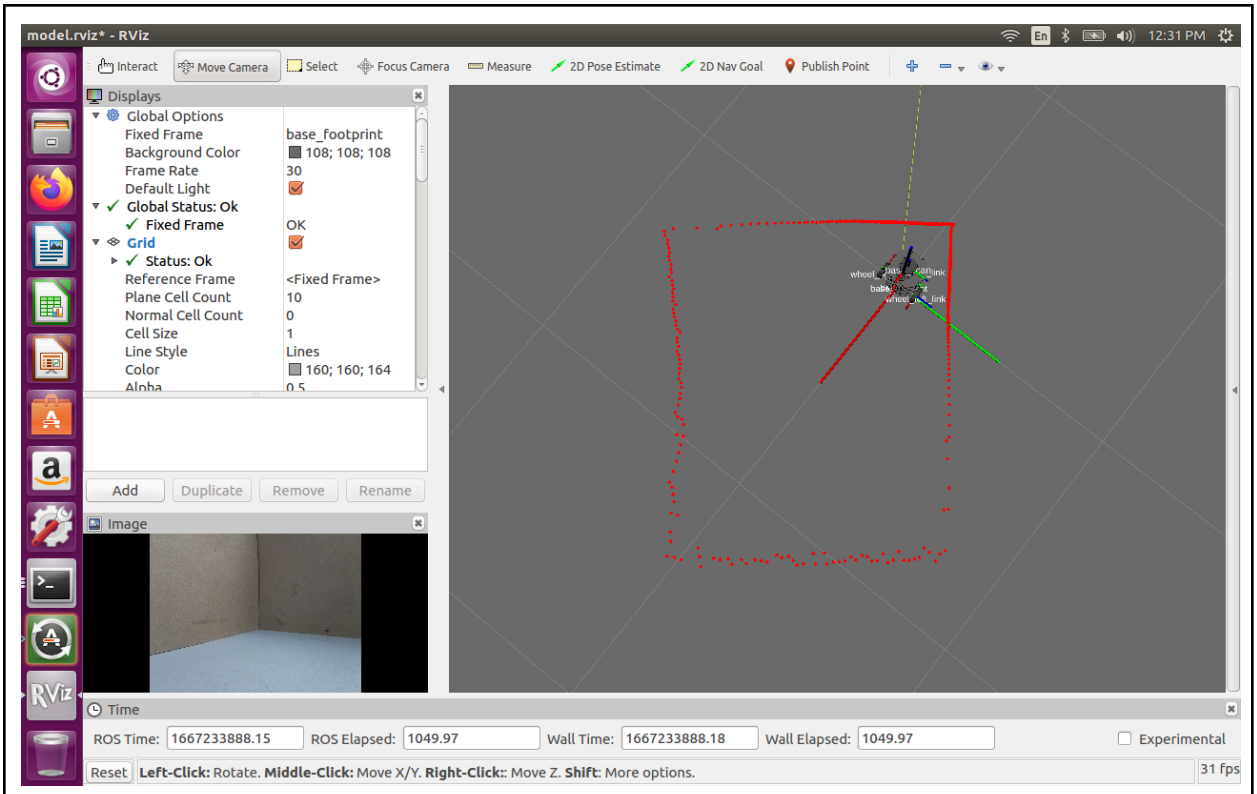


Figure 5: Step 2 T3 Robot placed at the northeast corner.

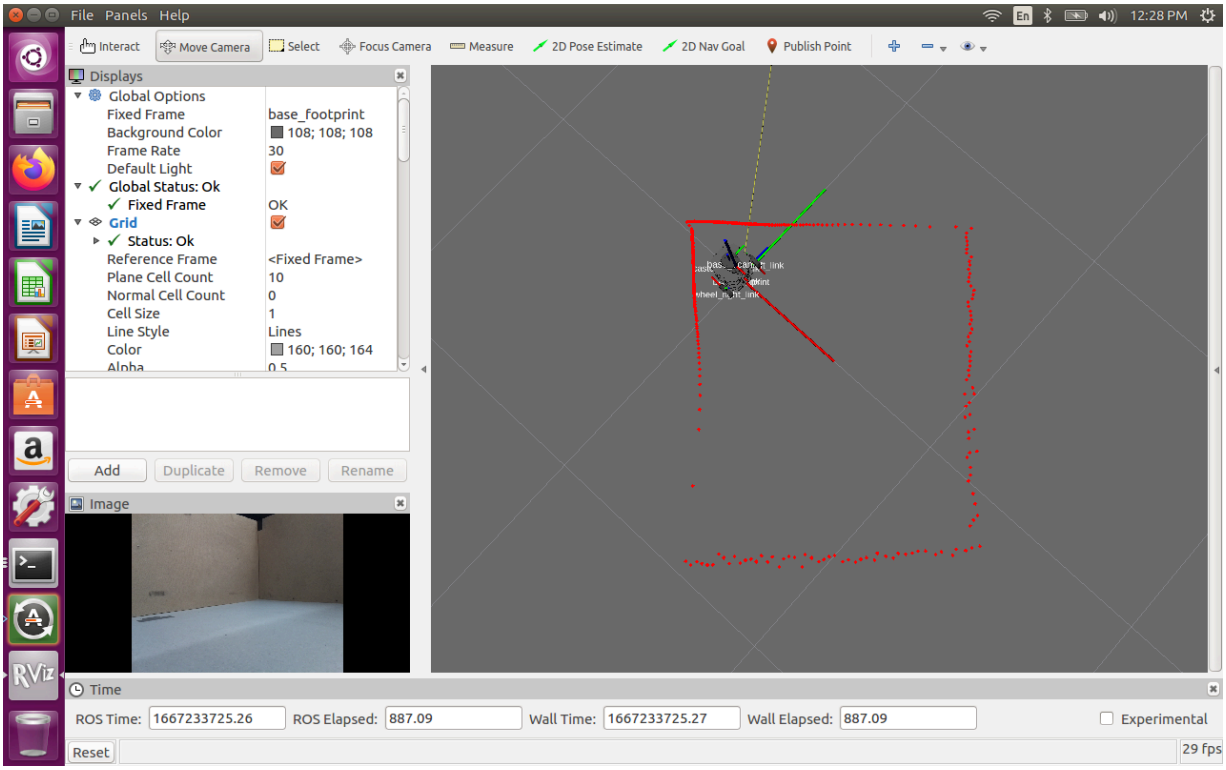


Figure 6: Step 2 T3 Robot placed at the northwest corner.

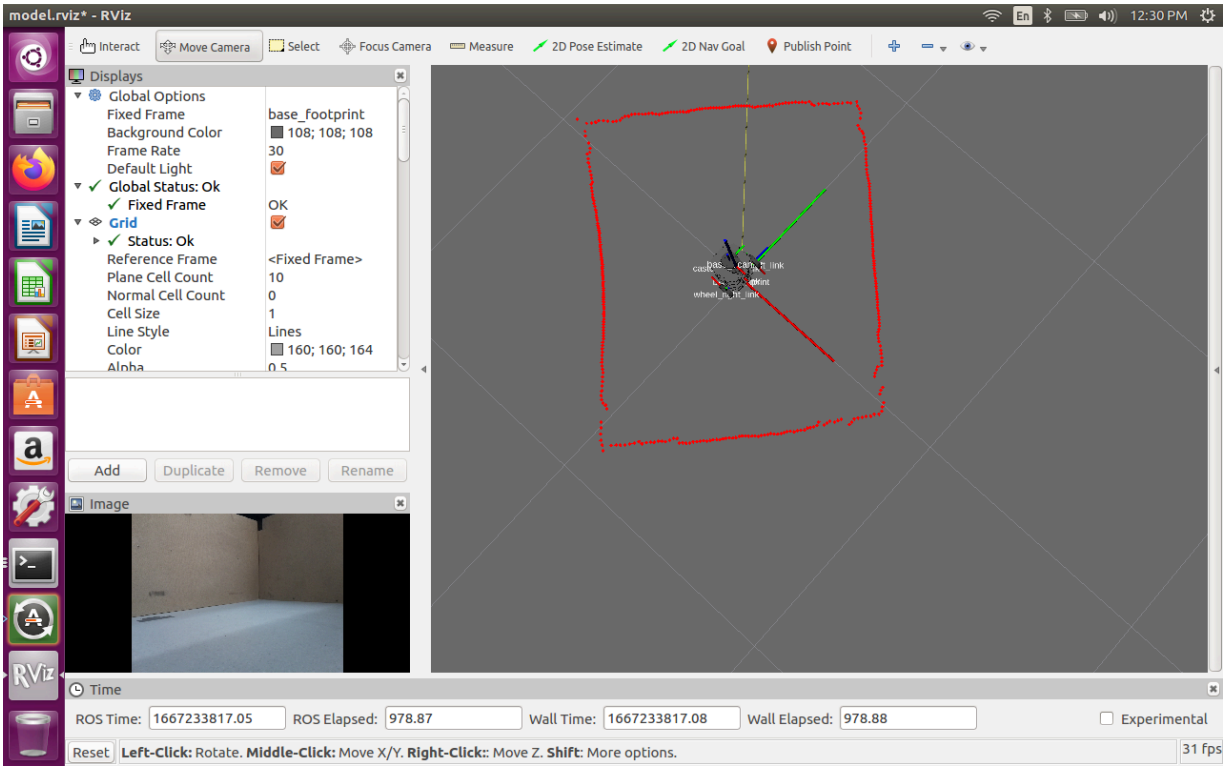


Figure 7: Step 2 T3 Robot placed at the center.

```
#!/usr/bin/env python
# license removed for brevity
#
# Example program that
# wanders,
# dml 2020
#
#

import math
import random

import rospy # needed for ROS
import numpy as np # for map arrays
import matplotlib.pyplot as plt

from geometry_msgs.msg import Twist      # ROS Twist message
from sensor_msgs.msg import LaserScan    # ROS laser msg




# ROS Topics
#
motionTopic='/cmd_vel' # turtlebot vel topic
laserTopic = '/scan'   # laser ranging topic

#global variable, to communicate with callbacks

gBumperLeft,gBumperRight= False, False # left/right close




# callback for the laser range data
# just sets the global Bump flags
#
def callback_laser(msg):
    '''Call back function for laser range data'''
    global gBumperLeft,gBumperRight

    gBumperLeft,gBumperRight=False,False
    numRays = len(msg.ranges) # total num readings
    radPerIndex = math.radians(360)/numRays

    width = int(numRays/6) # left/right bumper 'window'
    tooClose=0.5 # threshold for bumper to activate

    for i in range(0,len(msg.ranges)):
        #rule out bad readings first
        if not math.isnan( msg.ranges[i] ) and \
           not math.isinf( msg.ranges[i] ) and \
           msg.ranges[i]>0:

           # check for anything close left and right
           if msg.ranges[i]<tooClose:
               if i in range(0,width+1):
                   gBumperLeft=True

               elif i in range(numRays-width,numRays+1):
                   gBumperRight=True

    return


# wander_node -
# Moves frandomly until it detects a close surface
# then avoids it

def wander_node():
    '''continually move forward until a close surface is detected'''
    global gBumper

    # all ROS 'nodes' (ie your program) have to do the following
    rospy.init_node('Wander', anonymous=True)
```

```
def wander_node():
    '''continually move forward until a close surface is detected'''
    global gBumper

    # all ROS 'nodes' (ie your program) have to do the following
    rospy.init_node('Wander', anonymous=True)

    # register as a ROS publisher for the velocity topic
    vel_pub = rospy.Publisher(motionTopic, Twist, queue_size=10)
    # register as a subscribe for the laser scan topic
    scan_sub = rospy.Subscriber(laserTopic, LaserScan, callback_laser)

    # this is how frequently the loop below iterates
    rate = rospy.Rate(10) # Hz

    msg = Twist() # new velocity message
    msg.linear.x,msg.angular.z=0,0
    vel_pub.publish(msg) # stop all motors
    t = 0

    while not rospy.is_shutdown():
        # Simple OA strategy
        if t==0: # new velocities
            lvel = float(random.randint(0,5))/10.0
            avel = float(random.randint(-1,1))/10.0
            t = random.randint(1,40)

        msg.linear.x,msg.angular.z=lvel,avel

        if gBumperLeft or gBumperRight:
            msg.linear.x,msg.angular.z=-0.1,avel*10

        t = t-1
        vel_pub.publish(msg)
        rate.sleep()

    return

#
# This function is called by ROS when you stop ROS
# Here we use it to send a zero velocity to robot
# in case it was moving when you stopped ROS
#

def callback_shutdown():
    print("Shutting down")
    pub = rospy.Publisher(motionTopic, Twist, queue_size=10)
    msg = Twist()
    msg.angular.z=0.0
    msg.linear.x=0.0
    pub.publish(msg)
    return



#------------------------------MAIN  program----------------------
if __name__ == '__main__':
    try:
        rospy.on_shutdown(callback_shutdown)
        wander_node()
    except rospy.ROSInterruptException:
        pass
```

Figure 8: Original wander.py code

```python
def callback_laser(msg):
    '''Call back function for laser range data'''
    global gBumperLeft,gBumperRight

    gBumperLeft,gBumperRight=False,False
    numRays = len(msg.ranges) # total num readings
    radPerIndex = math.radians(360)/numRays

    width = int(numRays/6) # left/right bumper 'window'
    tooClose=0.2 # threshold for bumper to activate     #original 0.5

    for i in range(0,len(msg.ranges)):
        #rule out bad readings first
        if not math.isnan( msg.ranges[i] ) and \
           not math.isinf( msg.ranges[i] ) and \
           msg.ranges[i]>0:

            # check for anything close left and right
            if msg.ranges[i]<tooClose:
                if i in range(0,width+1):
                    gBumperLeft=True

                elif i in range(numRays-width,numRays+1):
                    gBumperRight=True

    return
```

```python
    while not rospy.is_shutdown():
        # Simple OA strategy
        if t==0: # new velocities
            lvel = float(random.randint(5,7))/10.0     # original (0, 5)
            avel = float(random.randint(-2,2))/10.0    # original (-1, 1)
            t = random.randint(1,40)

        msg.linear.x,msg.angular.z=lvel,avel

        if gBumperLeft or gBumperRight:
            msg.linear.x,msg.angular.z=-0.1,avel*10

        t = t-1
        vel_pub.publish(msg)
        rate.sleep()

    return
```

Figure 9: Adjusted wander.py code for test one.

```python
def callback_laser(msg):
    '''Call back function for laser range data'''
    global gBumperLeft,gBumperRight

    gBumperLeft,gBumperRight=False,False
    numRays = len(msg.ranges) # total num readings
    radPerIndex = math.radians(360)/numRays

    width = int(numRays/6) # left/right bumper 'window'
    tooClose=0.2 # threshold for bumper to activate     #original 0.5

    for i in range(0,len(msg.ranges)):
        #rule out bad readings first
        if not math.isnan( msg.ranges[i] ) and \
           not math.isinf( msg.ranges[i] ) and \
           msg.ranges[i]>0:

            # check for anything close left and right
            if msg.ranges[i]<tooClose:
                if i in range(0,width+1):
                    gBumperLeft=True

                elif i in range(numRays-width,numRays+1):
                    gBumperRight=True

    return
```

```python
    while not rospy.is_shutdown():
        # Simple OA strategy
        if t==0: # new velocities
            lvel = float(random.randint(1,3))/10.0     # original (0, 5)
            avel = float(random.randint(-2,2))/10.0    # original (-1, 1)
            t = random.randint(1,40)

        msg.linear.x,msg.angular.z=lvel,avel

        if gBumperLeft or gBumperRight:
            msg.linear.x,msg.angular.z=-0.1,avel*10

        t = t-1
        vel_pub.publish(msg)
        rate.sleep()

    return
```

Figure 10: Adjusted wander.py code for test two.

```python
    while not rospy.is_shutdown():
        # Simple OA strategy
        if t==0: # new velocities
            lvel = float(random.randint(1,3))/10.0     # original (0, 5)
            avel = float(random.randint(-2,2))/10.0    # original (-1, 1)
            t = random.randint(1,40)

        msg.linear.x,msg.angular.z=lvel,avel

        if gBumperLeft or gBumperRight:
            msg.linear.x,msg.angular.z=-0.1,avel*10

        t = t-1
        vel_pub.publish(msg)
        rate.sleep()

    return
```

```python
def callback_laser(msg):
    '''Call back function for laser range data'''
    global gBumperLeft,gBumperRight

    gBumperLeft,gBumperRight=False,False
    numRays = len(msg.ranges) # total num readings
    radPerIndex = math.radians(360)/numRays

    width = int(numRays/6) # left/right bumper 'window'
    tooClose=0.3 # threshold for bumper to activate     #original 0.5

    for i in range(0,len(msg.ranges)):
        #rule out bad readings first
        if not math.isnan( msg.ranges[i] ) and \
           not math.isinf( msg.ranges[i] ) and \
           msg.ranges[i]>0:

            # check for anything close left and right
            if msg.ranges[i]<tooClose:
                if i in range(0,width+1):
                    gBumperLeft=True

                elif i in range(numRays-width,numRays+1):
                    gBumperRight=True

    return
```

Figure 11: Adjusted wander.py code for test three.

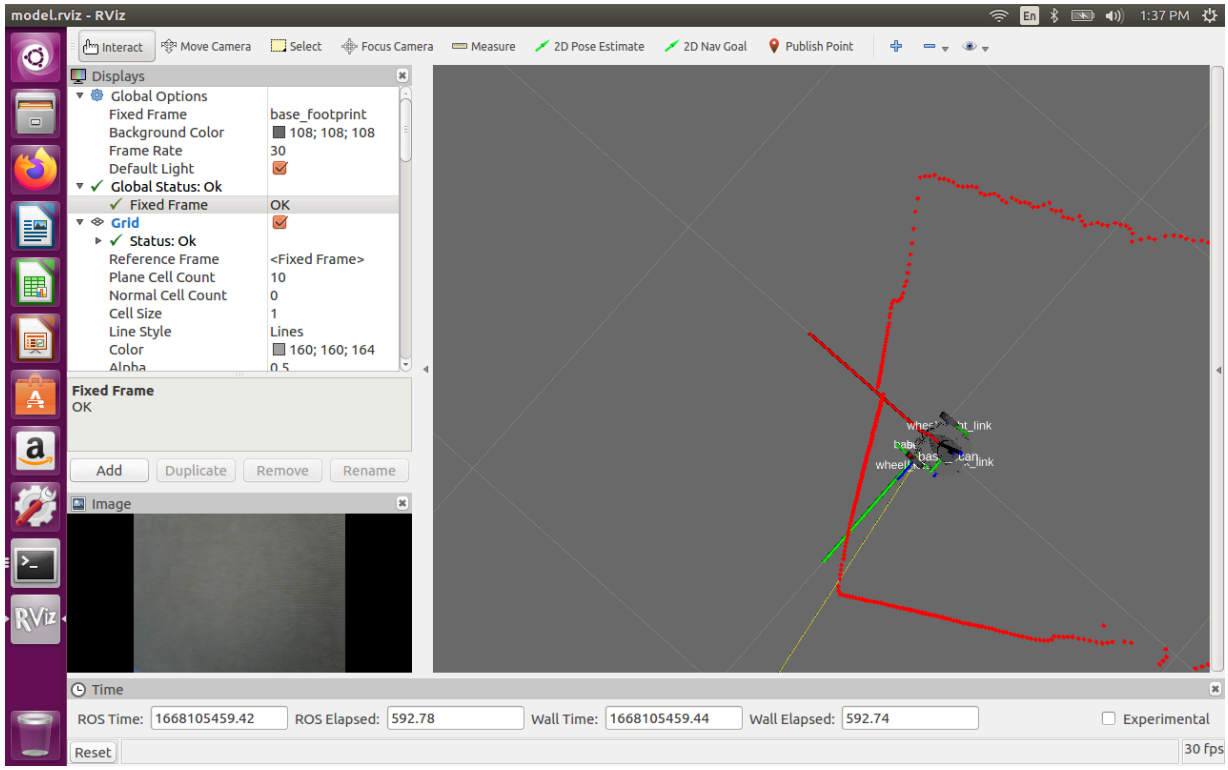


Figure 12: First screenshot of the T3 robot with adjusted wander.py file.

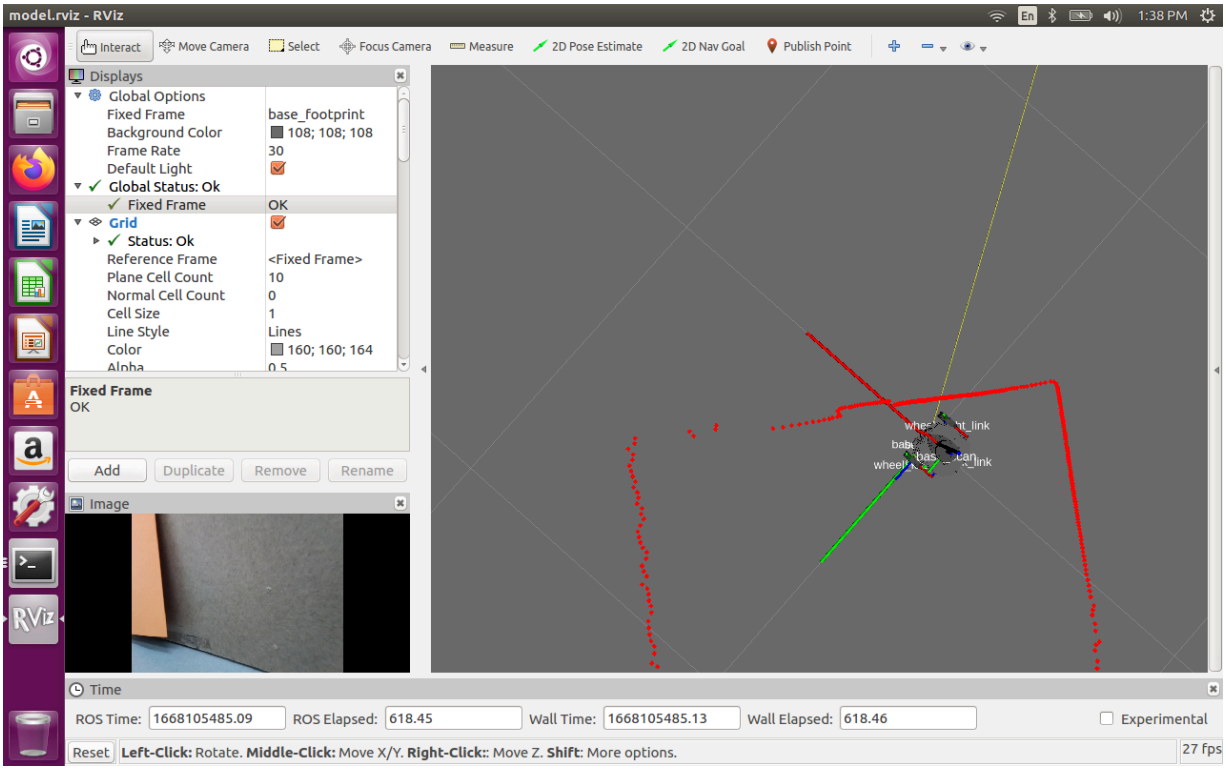


Figure 13: Second screenshot of the T3 robot with adjusted wander.py file.

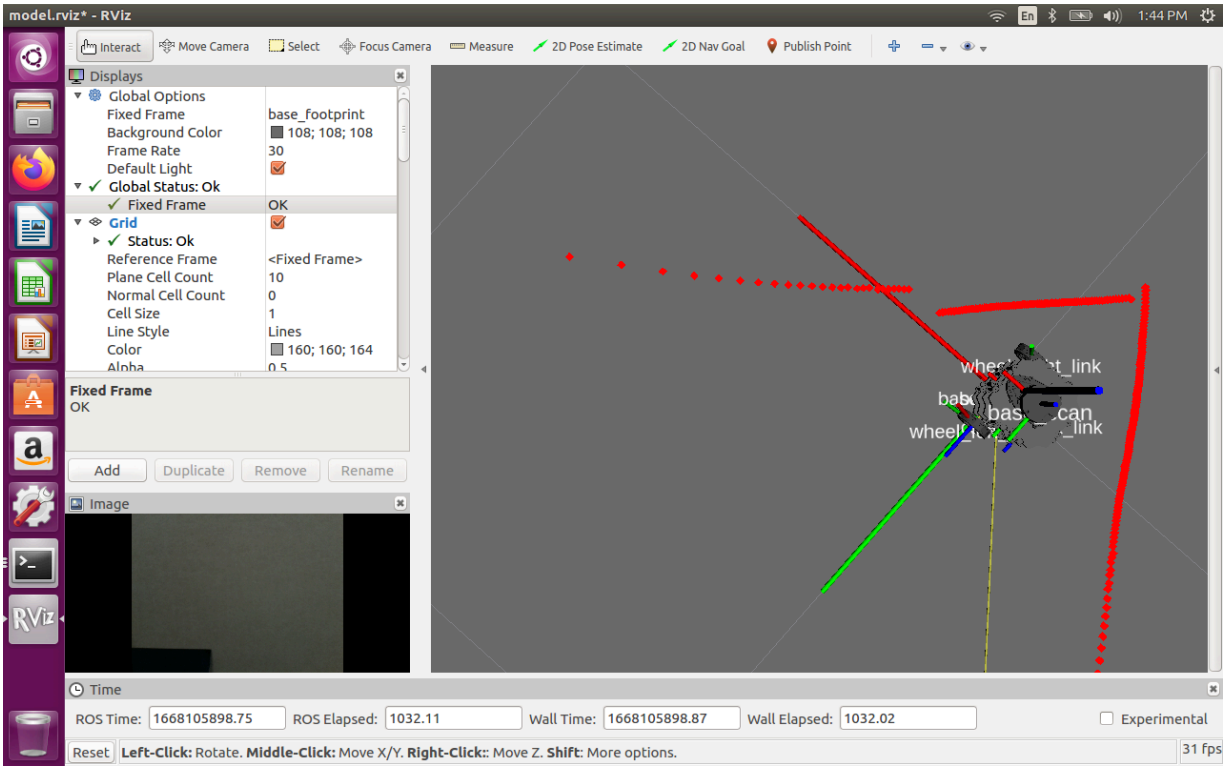


Figure 14: Third screenshot of the T3 robot with adjusted wander.py file.

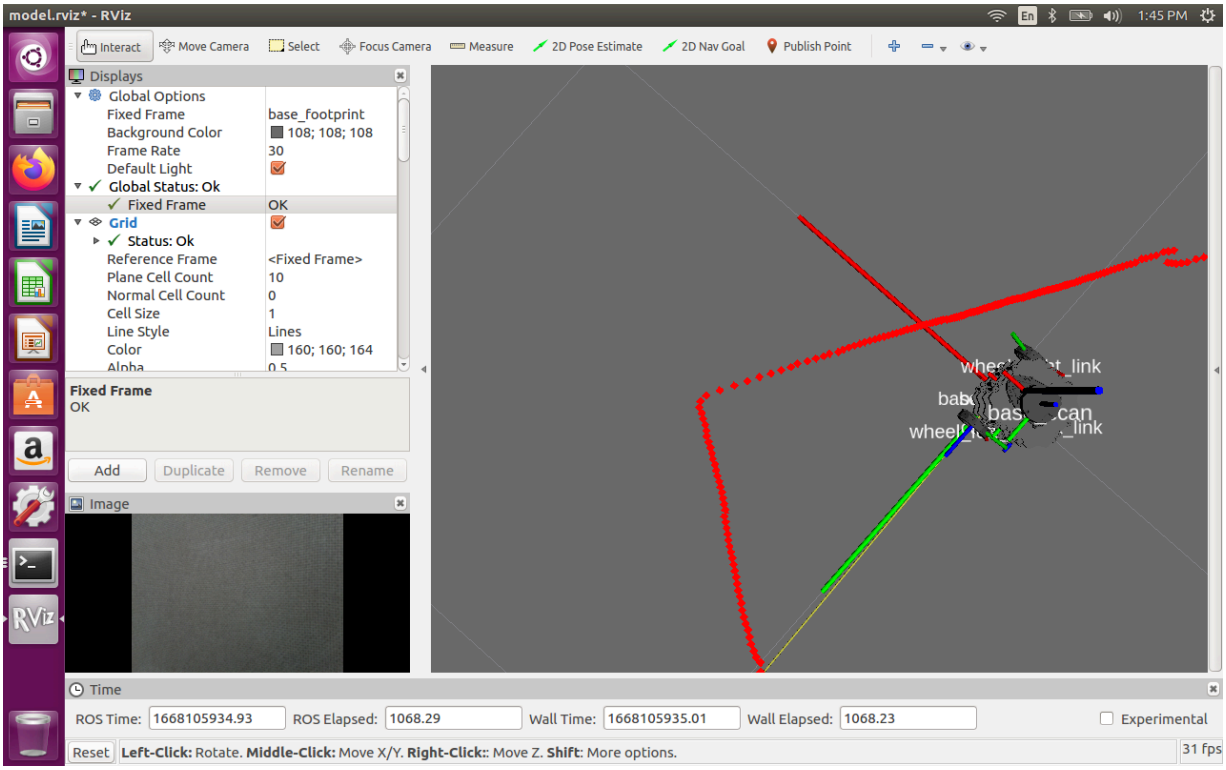


Figure 15: Fourth screenshot of the T3 robot with adjusted wander.py file.

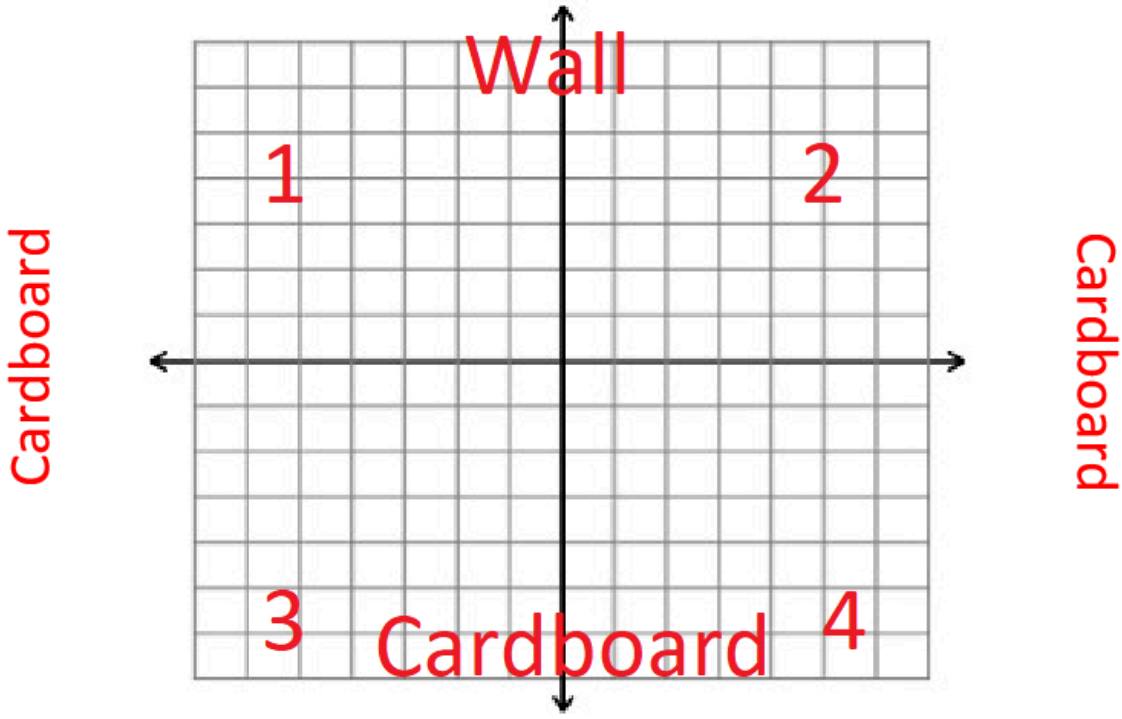


Figure 16: Grid of the world the T3 robot was placed in.

5.0 Summary

Based on the experiment done on the T3 robot, it can be observed that laser range sensing is definitely useful when robots track the environment around them. Although the laser points observed by the robot when placed in the square-like world were not perfect, a roughly accurate estimate was made by the robot. When changing the code in the wander.py file to enable the robot to sense the boundaries without hitting them, three tests were run to see what would be the most optimal. The third test seemed to be the most optimal and the most accurate, however some discrepancies still were present in the T3 sensing, as the boundary would be hit gently. Overall however, what can be observed is that with a small distance proximity of 0.3 m and a linear velocity of -1 to 3, and an angular velocity between -2 and 2, this is the closest the robot was able to get to with sensing a boundary and avoiding it.