Object detection and avoidance with ROSBots

fgpk20

December 2021

Contents

List of Figures

List of Tables

1 Problem Analysis

The challenge of getting a robot to avoid objects presents multiple issues. So as to enable a ROSBot to avoid objects a program can be written using the Robotic Operating System which takes the data produced by multiple sensors as input, interpreting it and then deciding on the actions required to continue forward and avoid any object which present themselves[1]. So in creating the program for the ROSBot to fulfil the task presented which is to create a program that allows it to navigate from one end of a room to the other without a clear line of sight with unknown objects in the way a few things are required. The first of which is that the robot needs to know in what direction relative to its current orientation its destination is so that it can move in the right direction, the second one is that of detecting when an object is blocking the path of the robot so that it is able to take the appropriate avoiding action. The first of these issues is relatively simple to overcome by setting the initial orientation of the robot to 0°and then when there is no object ahead of the robot ensuring the robot returns to this angle of travel

1.1 Challenges for object detection:

Objects can be detected through the array of different sensors present on a ROSBot as indicated in figure 1. The issue arises in determining which sensor input to use to detect an object so as to allow avoidance to take place. In the real world of the sensor present on the robot, the only one which would be able to detect all the obstacles which were present would have been the optical sensor however this would require more complex image processing. The other sensors available were the LIDAR and the IR sensors these however both have their own limitations. The one which affects them both is that the sensors are positioned at specific heights and as such only detect an object when it appears at that level. The other limitation is that both sensors are unable to detect specific materials.

The fact that both the LIDAR and IR sensors are found at different heights allows a GAZEBO environment to be used to simulate how in the real world objects may not be detected by one of the types of sensors. In the gazebo environment we are presented with a series of different objects these take the form of bins, coke cans, tables and bookcases as indicated in figure2 a-d, with each presenting their own issues. The bin, figure 2a in the simulation models allows for testing the object detection and avoidance mechanism by modelling an object which while being detected by LIDAR it is not detected by IR. The bin is able to behave as an object which can only be detected by the LIDAR due to there being a gap beneath the base of the bin due to its wheels. Whereas the bookcase, figure 2d, whilst behaving as a 4 sided object for IR due to the base of it being high enough the IR sensor however for LIDAR only 3 sides are visible, this creates problems as the if only the LIDAR were only used due to the bins there would also be the potential for a collision with the bookcase meaning that IR has to be included however there is also a potential problem as depending on

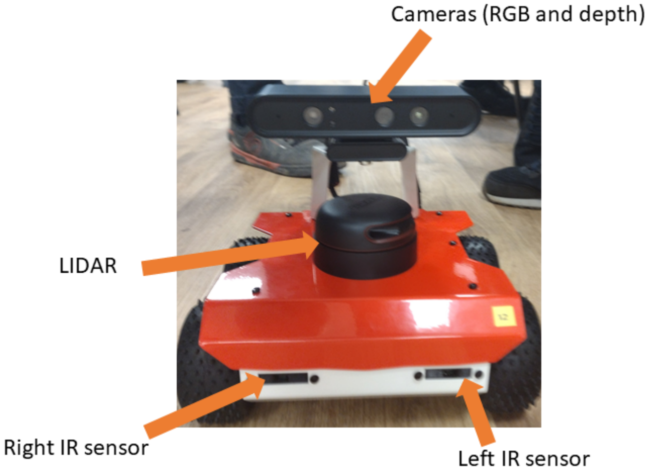


Figure 1: The location of the IR, LIDAR and camera sensors on a ROSBot

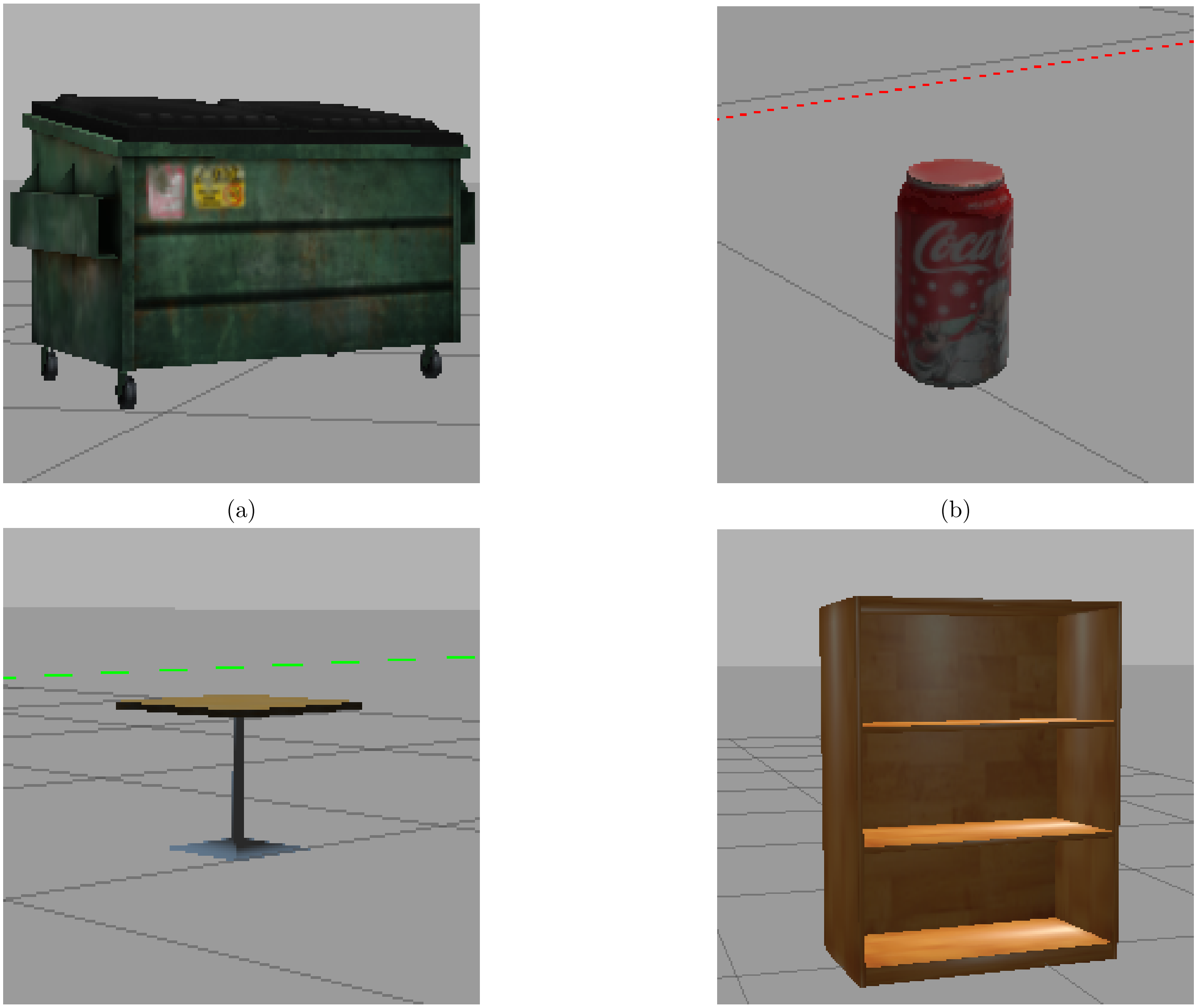
the action selection conditions may end up becoming confused by the two ends. The coke can and table, as shown in figures 2b,2c, behave in similar ways with their top being narrow and as such can result in them being seen by LIDAR but not by IR as it can be positioned between the two front IR sensors.

These different objects allow the three different situations to be dealt with the first being where the object can be both detected by IR and LIDAR, the second when the object is not seen by IR as happens when the surface of the obstacle has a metal coating, the final type of object is when it is not seen by LIDAR as happens when the obstacle is see-through (such as a plexiglass screen) When deciding what direction to turn when an object is detected there are multiple different ways. The simplest one would be that when an object is detected the robot turns either left or right in a very simple state machine, this however has the potential of getting the robot trapped in any form of robot trap. By randomizing this decision to turn left or right when an object is encountered while it may momentarily get trapped the random selection of the direction which it turns when it encounters an object would result in it escaping the trap, in our case of trying to navigate from one end of the room to the other this would mean that the total distance travelled would be less than if it always tuned in the same direction if an object was encountered. The action selection mechanism which is being used here results in the robot turning in the opposite direction to that which an object is detected however as the robot is constantly checking to see if the object is still there this means that once the object is no longer presenting an obstruction it will turn back in the direction of the target thus avoiding robot traps.

The original intention for the program was for individual nodes which processed the data from the sensors two feed into a node which decided the action selection which then in turn feed into a node which outputted the control as indicated in figure 3, however due to the complexity of the system it was felt that it would be easier to create a system which instead of publishing and then subscribing to these individual things it would be easier to create a single node witch did all of this with the processing of data happening in functions inside the single node. Whilst the former approach to the problem may be more adaptable when it comes to including additional features, as the original code structure called for more information than whether or not an object was there or not it was easier to create a single node.

2 Software architecture

The program which was developed subscribes to the front IR sensors(/range/fl,/range/fr) and the LIDAR sensors (/scan). An initial iteration of the



(a) (b) (c) (d)

Figure 2: Obstacles present in the simulations

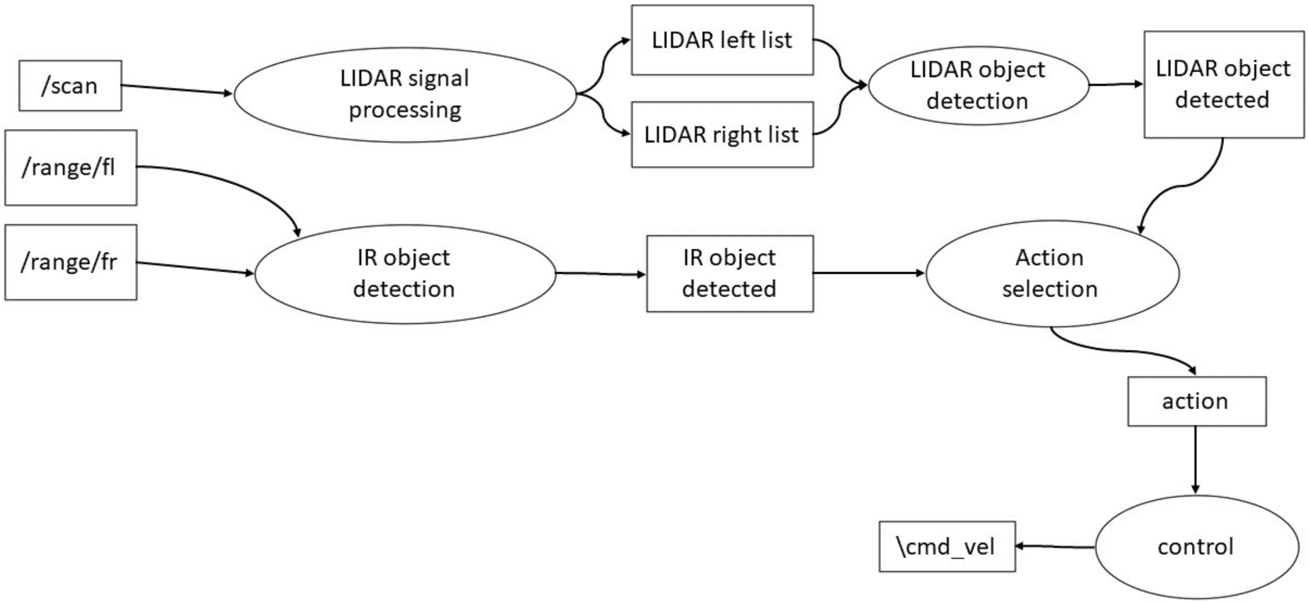


Figure 3: Original plan for node and topic structure which was then recreated inside the /single_project_node instead of as individual nodes and topics

program was designed to process this data in separate nodes before publishing it into a node which would carry out the action selections, However, the method which has been presented in figure 4 was used instead as data from the sensor nodes was still being called in the action selection process and so to simplify this the processing of the data was moved into the same node as the action selection (/single_project_node).

The inner workings of the node /single_project_node can be described by figure 3. The node subscribes topics where the IR sensor data and the LIDAR sensor data is published, the IR data for both the left and right sensor is passed through a function that determines whether or not an object is present ahead, whereas the LIDAR data is passed through a function which splits it into a list of ranges for the left-hand side and a list for the right-hand side of the robot this is then passed through another function which determines whether or not an object is ahead of it. The action selection part of the program considers initially whether or not there is an object ahead according to the IR data followed by whether or not there is an object ahead according to the LIDAR data, this is done in this order as the IR is being treated as a short-range sensor and the LIDAR of more of a long-range planning sensor if either of the sensors detects an object ahead of the robot then it turns away from it whilst moving forward this continues to happen until the object is no longer ahead of the robot. Once there is no object detected ahead of the robot then the robot attempts to realign its self with the direction that it is attempting to travel in and move forward.

3 Software implementation:

3.1 Object detection functions:

The two front IR sensors as indicated in figure 5 only detect along a specific line and also have a maximum range, as such the function in the code which detects when there is an object ahead of the robot it checks if the object is within a certain threshold distance and if so it returns True otherwise it returns False.

The LIDAR sensor provides a list with the ranges with each item in that list being the distance two the nearest object at different angles. This is then split in two lists on for the left side and on for the right side of the robot as indicated

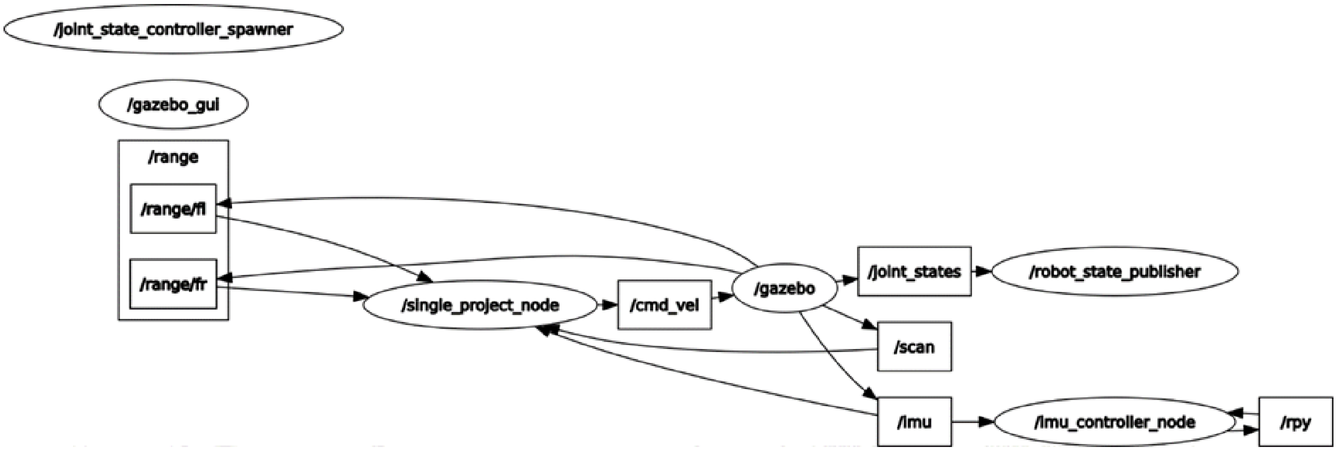


Figure 4: Rqvis graph showing the node and topic architecture which was used

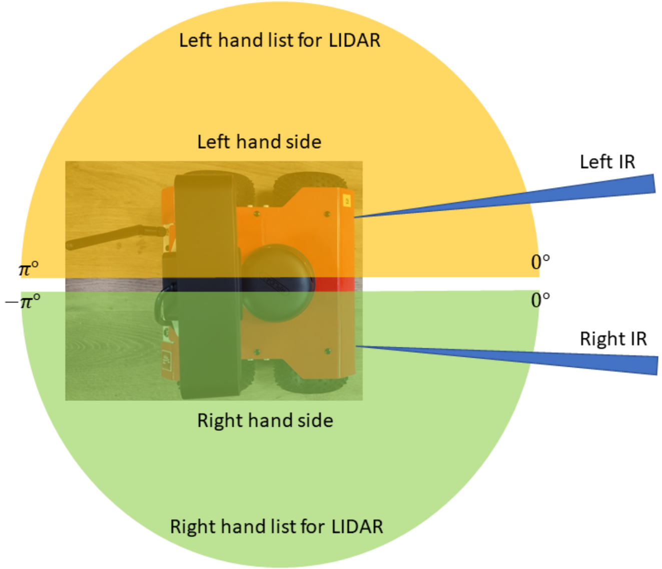


Figure 5: The field of view around the robot for the front IR and the LIDAR

on figure 5, these lists are then put through the object detection function for the LIDAR sensor. The object detection function considers if there is an object within a certain range of the robot directly ahead this is done by setting a minim range to the object, however by only considering the detection point directly ahead objects which are potential in the path of the robot are necessarily considered as such a greater range of angles need to be considered for object detection. This is done through the use of the equation 1 which allows the distance of an object form the center line to be determined as indicated in figure6, by determining the distance which an object is from the center line if it is below a specific threshold then True is returned.

$$x = r \sin \theta \tag{1}$$

3.2 Movement determination function:

The action selection process, as outlined in figure 7, is implemented by initially checking if either the left or right IR detects an object and then turning based weather the left of right IR has a longer range till the object is detected, the robot continues to turn until there is no longer an object detected ahead with in the threshold range and then it heads straight for a second before it can attempt to head straight ahead again. The IR is the first condition which checked due to it being treated as the sorter range sensor. If the IR detects an object as the sensors are slightly indented, as seen in figure 1, a counter is created so that if no other objects are detected by the LIDAR or IR then before it turns back to the direction of the end of the room it travels forward for a period so that it does not get court on the edge of the object.

The LIDAR then checks if there is an object to the left and if there is it turns the robot to the right until the object is no longer seen at the same time as the robot moves forward. Once there is no object on the left the LIDAR checks to see if there is an object on its right and if there is it will turn to its left until the object is no longer seen whilst still moving forward. This however has the potential to lead to indecision as once there is no longer an object on the left side of the robot then it will check to see if there is now an object on the right hand side and if so it will turn to the left at which point it could encounter an object at which point it will turn back potentially encountering the object on the right hand side again as indicated on figure 8. To overcome this indecision a flag in raised where by if one of the two conditions has just been satisfied and now the other one is then it will continue to turn in the same direction instead of back.

When there is no object detected ahead of the robot the robot orientates its self back to the orientation which it was placed at the beginning of the run by seeing it the angle of the robot is greater than or less than 0 and turning in the shortest direction back to the direction as which it was originally placed by taking data form the /imu as to what this angle is. When there are no object detected and the robot is pointed in the right direction the robot moves forward at a constant rate. To determine the thresholds and any constants that were used in the code as well as speeds at which the robot moved forward and turned testing was carried out in a known simulation environment out lined bellow.

4 Testing approach:

The software was tested initially in a known world in the virtual simulation environment GAZEBO which was provided to us, figures 9. Through the use of a this known world enabled the trialing different bits of software so as to determine their deficiencies allowing them to be rectified before running the software on the physical ROSBot or in the unknown world in the simulation environment. The use of the known world in the virtual known environment also allowed for the trilling of different threshold values for the different sensors enabling these values to be tuned to around the optimum value. The software was finally tested on both the physical ROSBots with previously unseen obstacles provided, as well as an unseen simulation environment provided to us.

So as to determine the efficiency of the object avoidance software which had been created both the time the robot to travel form one end of the

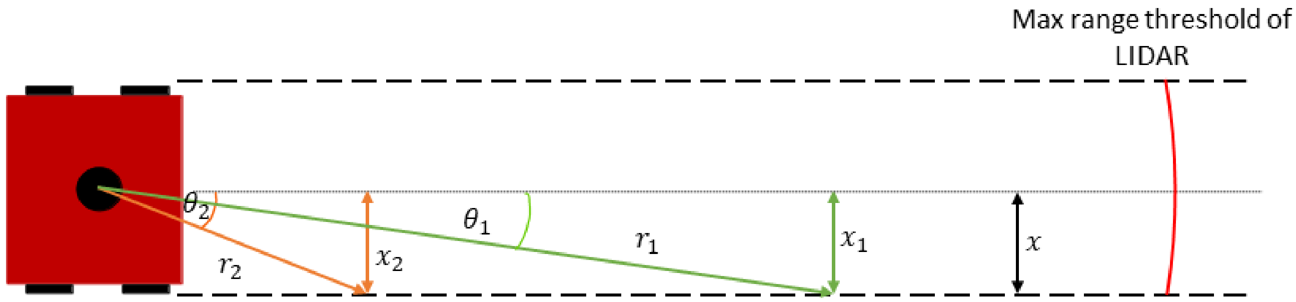


Figure 6: How equation 1 is implemented so that the robot is able to detect object which are ahead of it using a greater number of sensor points

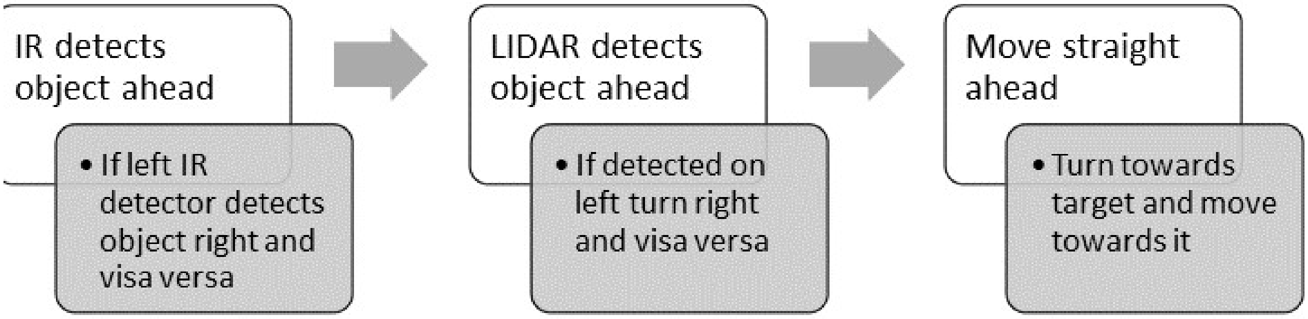

IR detects object ahead
• If left IR detector detects object right and visa versa
LIDAR detects object ahead
• If detected on left turn right and visa versa
Move straight ahead
• Turn towards target and move towards it

Figure 7: The process used to determine what action is undertaken by the robot based on the sensor input

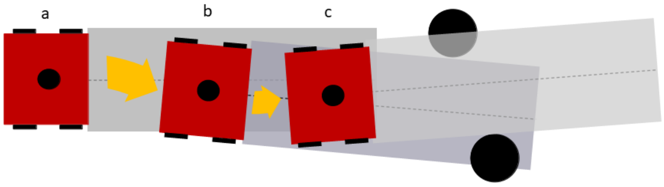


Figure 8: Example of a situation where dithering can sometimes be seen. a) object detected on the left side of the robot path according to the LIDAR data and so turns to the right, b) object no longer detected on the left but object detected on the right side of the robot path according to the LIDAR data and so turns to the left, c) object no longer detected on the right but detected on the left side of the robot path according to the LIDAR data and so turns to the right resulting in a collision occurring with the obstacles

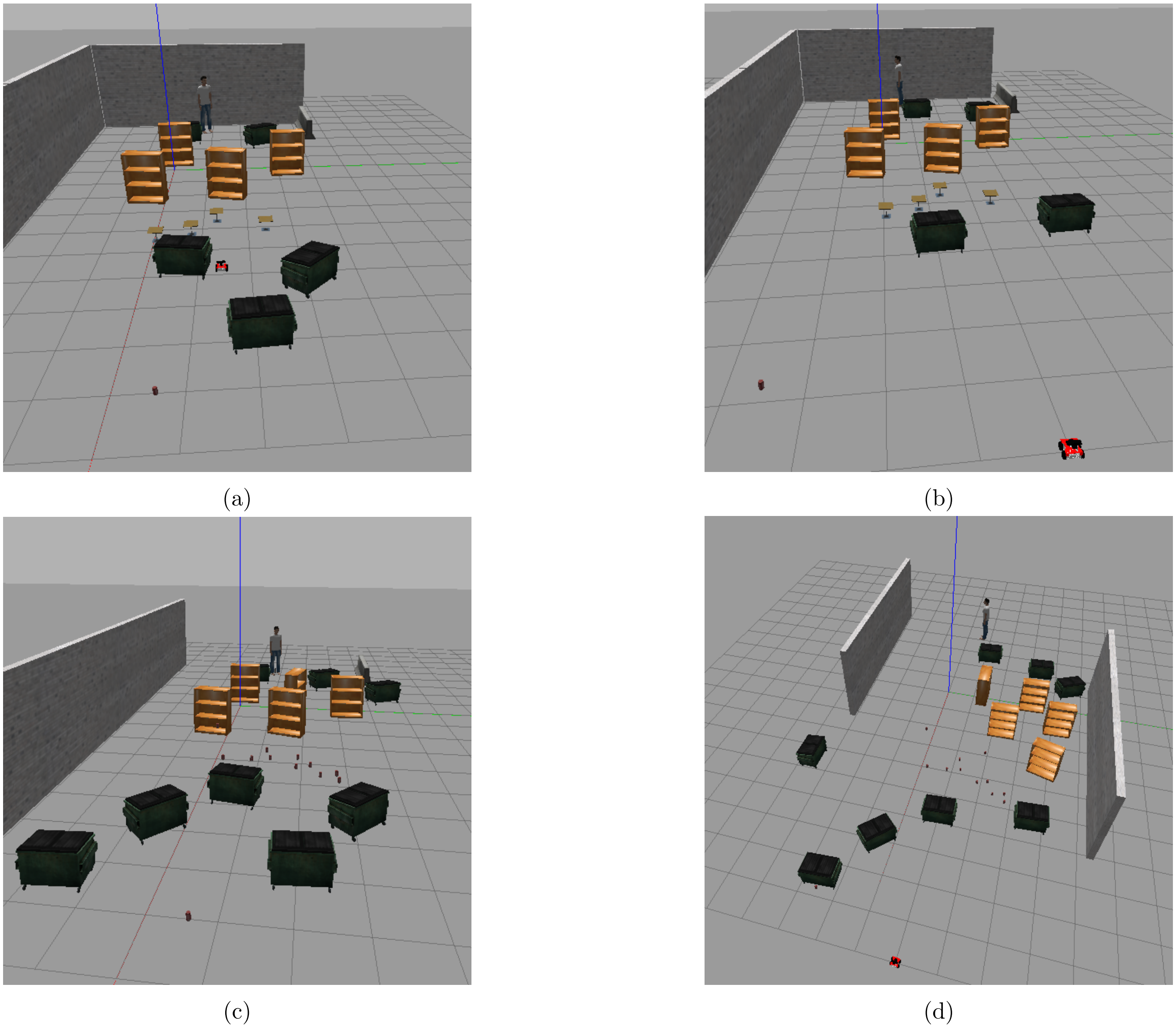

(a)
(b)
(c)
(d)

Figure 9: Simulation environments, a) and b) known worlds, c) and d) unseen worlds

simulation environment to the other and the distance which it traveled were calculated. So as to determine the distance traveled in a specific time there speed at which the robot was traveling was traveling as given in /cm_vel was multiplied by the inverse of the ROSRate to give the distance traveled in the given interval. The distances travlled in these intervals were then summed to gather to get the total distance that was travelled by the ROSBot. These distances can be compared to those that are calculated through the use of the position given by /odom in the gazebo simulation environment with the total distances traveled to be found to be very similar.

5 Testing Results:

5.1 Testing in the known simulation environment:

The testing in the known simulation allowed for objects to be moved around so as to trial differences in designs in the code. Starting with the IR sensor the initial challenge is to get the robot round a solid object when it is detected. This was done by moving a book case in front of the robot and testing the code, the initial part of testing was to check that the object detection and action selection criteria worked, which they did with the action selection criteria being to turn in the direction where the distance to the robot according to the IR sensor was greatest. When doing this it was found that when the robot reached the end of the book case and when it tries to realigned with the direction of travel, as the sensors are slightly indented, it would get caught on the edge of book shelf as at shorter ranges the IR detection beam is inside the width of the robot as is seen in figure 10. So as to prevent the robot form getting caught on the edge of the book self it was decided that after it had detected the object for the last time it would continue in a straight line be for attempting to realign with the intended direction of travel in the testing of this the optimal time was decided through trial and error, as given in table 1, with the threshold ranges withing with the object would be detected affecting the time which the robot moved forward. The time which the robot moved forward was decided through creating a counter which was a multiple of the rosrate. Whilst experimenting with the parameters it was decided that it would be more efficient if the robot was to move forward at the same time as detecting the object, rather than stop and turning each time an object is detected.

If only the IR sensor is used while it may be able to deal with most situations, as it is unable to detect some objects in the simulation environment, such as the bins which it pushes around the world. The other issues is that when a confined space is encountered in the case of a robot trap if the detection range is small the robot may collide with the with objects in the simulation world if it moves whilst turning, one potential method to counter this would be to have a scaling system where by the closer the obstacles are to the robot the slow the speed to the point whereby below as specific threshold distance to the obstacle it would no longer move forward, a potential method to decide the speed for travel would be the code in figure 11. When looking at the data in table 1 it can be seen that if the object is only detected below 0.2 m the robot collides with the obstacle this is due to the IR sensor being indented and thus angled and at a range of 0.2 m the IR beam is within the width of the potential robot path thus any obstacle detected will catch the edge of the obstacle, as shown in figure 10.

The LIDAR detection method was tested by placing the robot in front of the bin and the value of minimum value for x where the object is not present, as indicated in figure 6, was varied so as to see for what values of x the robot is able to pass the bin without a collision occurring. The threshold detection range as well as the speed at which the robot turns was also varied, the results from this are presented in table 2. From table 2 the minimum value for x can be seen to be around 2 which is approximately the width of the robot so one would have assumed that it would need to be approximately half of that as the distance form the center of the robot too the front corners is close to 0.2 m the value of c has to be closer to this so that as it navigates the corner it leaves enough space to rotate back to the direction of travel, for a similar reason the length of the list

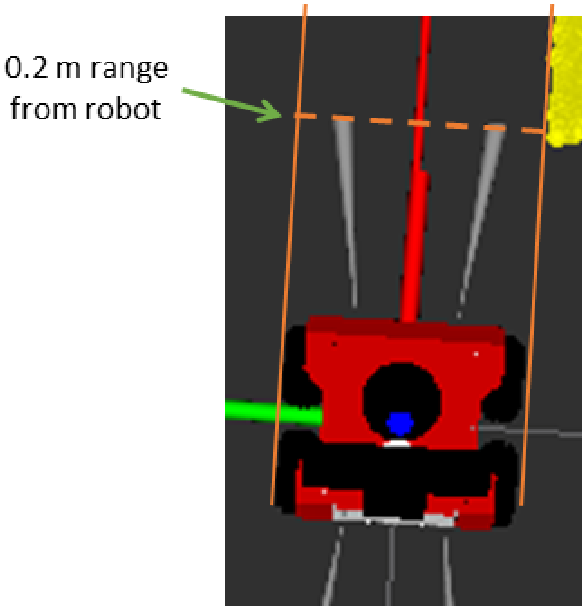
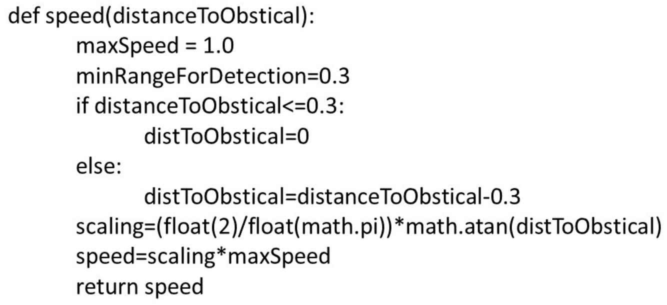


Figure 10: rvis graph showing how IR detection at 0.2 m from robot is with in the width of the robot

Max detection threshold (m)	Time robot continues to move forward (s)	Absolute angular velocity when object detected	Speed when object detected	Speed when moving forward when object no longer detected	Collision
0.7	3	0.5	0.1	0.2	Yes
0.7	4	0.5	0.1	0.2	No
0.3	4	0.5	0.0	0.2	No
0.3	4	0.5	0.05	0.2	sometimes
0.3	4	0.5	0.1	0.2	Yes
0.2	4	0.5	0.0	0.2	Yes
0.2	4	0.5	0.0	0.1	Yes
0.5	4	0.5	0.0	0.2	No
0.5	4	0.5	0.1	0.2	No
0.4	4	0.5	0.0	0.2	No

Table 1: Testing IR detection and its parameters.



```
def speed(distanceToObstical):
        maxSpeed = 1.0
        minRangeForDetection=0.3
        if distanceToObstical<=0.3:
                distToObstical=0
        else:
                distToObstical=distanceToObstical-0.3
        scaling=(float(2)/float(math.pi))*math.atan(distToObstical)
        speed=scaling*maxSpeed
        return speed
```

Figure 11: A potential function to regulate the speed when and object is detected based on $\textit{speed scaling factor} = \frac{2}{\pi}arctan$(dist to object) as this gives tends to 1 as the robot moves away form the object and 0 as it moves towards the object

being considered is upto 90 degrees to the direction of travel instead of 45 degrees to the direction of travel. If the speed that the robot rotates back to the objective is to fast it can result in the object being detected on both the left and right hand sides meaning that even if it is on the right hand side of the robot condition for it being on the left hand side is satisfied and thus it will rotate to the right, to prevent this the software was modified so as to check whether the range to the object was greater in the left list then in the right list.

The IR functions and the LIDAR functions are then combined so as to allow for the system to be optimized based on the constants the variables which were trialed when considering only the LIDAR and only the IR tables 1 and 2, to test the effectiveness of the different parameters the total distance traveled and the time taken where compared in table 3 for the simulation worlds sown in figures 9. It was found that by using shorter maximum detection thresholds the distance traveled by the robot was generally less, as at longer ranges the robot would be avoiding an object and detect an object in its vicinity which is not in the path to its objective causing it to take evasive action even though this is not necessarily necessary as it may not be presenting an obstacle causing is to travel a greater distance parallel to the direction of intended travel. It was also found that by reducing the speed at which the robot traveled whilst this increased the time for the robot to get from one end of the world to the other it prevented the sensors getting confused due to the speed of turning, as when turning quickly the robot would miss the obstacle appearing in one sensor input on one side and not the other which happened when the robot turned at a slower rate. By comparing the results for the distance traveled in each case as derived form the /odom topic as published by gazebo and the distance calculated form the speed of the robot it can be seen that they are close enough for either method to be used and the results from the speed to be relied upon in the real world to give the distance traveled.

5.2 Testing in unseen simulation environment and on physical robot:

In the unseen simulation environment the robot was run using the parameters laide out in table 3, in both cases unseen simulation environments the robot was able to navigate form one end of the map to the other without colliding with any obstacles. The hardest obstacle for the robot to avoid was the coke cans as this can lead to the indecision as has been described previously been discussed and was shown in figure 8, the inclusion of the flag prevented this form occurring. In the real world the when testing the robot had the greatest problems when it came to considering objects such as the cone where the base was wider than the detectable part of the object meaning that although the robot attempts to avoid the part which is visible to both IR and LIDAR it collides with the base. This was also seen with the supports which were holding up the plexiglass screens.

Minimum value of x (m)	Max detection range (m)	Absolute angular velocity when object detected	Speed when object detected	Max angle of LIDAR data considered	Angular velocity to return to direction of objective	Speed when turning back to direction of objective	collision	notes
Obstacle being considered is the bin								
0.2	2.0	0.5	0.1	45°	0.5	0.0	Potentially	Can results the turning to the right even if object is on the right
0.2	2.0	0.5	0.1	45°	0.5	0.1	Potentially	Can results the turning to the right even if object is on the right
0.2	2.0	0.5	0.1	45°	0.25	0.1	Yes	
0.2	2.0	0.5	0.1	90°	0.25	0.1	No	
0.15	2.0	0.5	0.1	90°	0.25	0.1	Yes	
Containing condition checking if LIDAR detection on the left is longer than that on the right if object is detected by LIDAR								
0.2	1.0	0.5	0.1	90°	0.5	0.2	No	
0.2	0.5	0.5	0.1	90°	0.5	0.2	No	
0.2	0.3	0.5	0.1	90°	0.5	0.2	Yes	
0.2	0.3	0.5	0.0	90°	0.5	0.2	No	
Obstacle being considered in the book case								
0.2	1.0	0.5	0.1	90°	0.5	0.2	No	
0.2	0.7	0.5	0.1	90°	0.5	0.2	Yes	
0.2	0.7	0.5	0.0	90°	0.5	0.2	NO	
0.2	0.5	0.5	0.0	90°	0.5	0.2	Yes	

Table 2: Testing IR detection and its parameters.

IR threshold (m)	LIDAR threshold (m)	Absolute angular velocity when object detected IR	Speed when object detected IR	Absolute angular velocity when object detected LIDAR	Speed when object detected LIDAR	Absolute angular velocity turning back to direction of objective	Speed when turning back to direction of objective	Speed otherwise	Time taken (s)	Distance traveled form /odom (m)	Distance traveled form speed of robot (m)
Sim world as presented in figure 9a											
0.3	0.7	0.5	0.0	0.5	0.0	0.5	0.2	0.2	96	14.97	15.11
0.5	1.0	0.5	0.1	0.5	0.1	0.5	0.2	0.2	82	14.95	14.92
0.5	1.0	0.5	0.0	0.5	0.1	0.5	0.2	0.2	86	14.95	14.84
0.5	1.0	0.5	0.1	0.5	0.1	0.25	0.1	0.2	93	14.40	14.37
0.7	2.0	0.5	0.1	0.5	0.1	0.25	0.2	0.2	87	15.78	15.65
0.7	1.5	0.5	0.1	0.5	0.1	0.25	0.2	0.2	98	17.80	17.85
0.7	1.5	0.5	0.1	0.5	0.1	0.25	0.0	0.2	127	14.37	14.60
0.7	1.5	0.5	0.1	0.5	0.1	0.25	0.1	0.2	87	14.00	14.01
Sim world as presented in figure 9b											
0.5	1	0.5	0.1	0.5	0.1	0.5	0.2	0.2	82	15.19	15.18
0.5	1	0.5	0.0	0.5	0.1	0.5	0.2	0.2	82	15.19	15.18
0.3	0.7	0.5	0.0	0.5	0.0	0.5	0.2	0.2	84	15.24	15.41
Test world figure 9c											
0.5	1.0	0.5	0.1	0.5	0.1	0.25	0.1	0.2	99	14.53	14.54
0.5	1.0	0.5	0.1	0.5	0.1	0.5	0.1	0.2	99	15.58	15.59
Test world figure 9d											
0.5	1.0	0.5	0.1	0.5	0.1	0.25	0.1	0.2	88	14.38	14.35
0.5	1.0	0.5	0.1	0.5	0.1	0.5	0.1	0.2	92	14.15	14.13
0.5	1.0	0.5	0.0	0.5	0.0	0.25	0.1	0.2	99	14.34	14.43

Table 3: Results form using both the IR and LIDAR sensors in the different simulation environments with different parameters

6 Conclusion and further work:

The software which has been created and implemented here is sufficient for the task at hand, however there are a few object which if encountered due to the sensors being used the robot is unlikely to react to them. The primary one which would need to be considered is that of an object such as a as plank which is below is below the level of the both the IR and LIDAR sensors, so as to prevent a collision additional sensors would be needed which can detect objects which are at have there maximum high at lower levels. This problem was sometimes seen when a robot encountered a cone during the ROS challenge as while the body of the cone was visible to both the IR and LIDAR sensors is base was considerably wider meaning that while the ROSBot may react to the cone as it is unable to see the wider base a collision could still occur. By including the data form the camera more specifically form the depth cameras there detection would be enabled as the signal could be processed in such a way so as to determine that the area ahead of the robot is free objects anywhere ahead of it, this also has the potential to enable planning of the route that the robot can take to would take to its target as.

An alternative method which would allow planning would be the implementation of Simultaneous localization and mapping (SLAM)[2]. By mapping its environment the robot is able to plan and the optimum route around objects environment, however while the map the robots environment allows for planning of routes around static objects this becomes a challenge when there are moving objects in the simulation such as other robots or humans. So as to implement SLAM successfully the structure of the software would need to be altered to more of a behavior orientated design as well as being split across multiple different nodes so to allow for easier adaption of the code when things change.

References

[1] M. Quigley, B. Gerkey, and W. D. Smart, *Programming Robots with ROS: a practical introduction to the Robot Operating System.* " O'Reilly Media, Inc.", 2015.

[2] C. Cadena, L. Carlone, H. Carrillo, *et al.*, "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age," *IEEE Transactions on robotics*, vol. 32, no. 6, pp. 1309–1332, 2016.

7 Code Appendix

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

import rospy
import sys
import math
from nav_msgs.msg import Odometry
from sensor_msgs.msg import Imu,Range, LaserScan
from geometry_msgs.msg import Point, Twist
from rosgraph_msgs.msg import Clock
from math import atan2
from std_msgs.msg import Int32, Bool
from tf.transformations import euler_from_quaternion

#defining variables
Lrange = 10
Rrange = 10
absRDist=0.0
absLDist=0.0
maxLrange=0.0
maxRrange=0.0
angleToNormal=0.0
LeftLIDARlist=[]
RightLIDARlist=[]
incrimentAngle=0.0
currentTime=0
xVel=0.0
yVel=0.0
odDist=0.0
x_new=0.0
y_new=0.0
x_old=0.0
y_old=0.0

#proccesing functions
def leftRange(msg):
  global Lrange
  global absLDist
  global maxLrange
  Lrange=msg.range
  maxLrange=msg.max_range
  absLDist=Lrange/maxLrange

def rightRange(msg):
  global Rrange
  global maxRrange
  global absRDist
  Rrange=msg.range
  maxRrange=msg.max_range
  absRDist=Rrange/maxRrange

def procImu(msg):
  global angleToNormal
  rot_q=msg.orientation
  (roll,pitch,angleToNormal)=euler_from_quaternion([rot_q.x, rot_q.y, rot_q.z, rot_q.w])
  global xVel
  global yVel
  xVel=msg.linear_acceleration.x
  yVel=msg.linear_acceleration.y

def LIDARprocess(msg):
  global LeftLIDARlist
  LeftLIDARlist = []
  global RightLIDARlist
  RightLIDARlist =[]
  global incrimentAngle
  incrimentAngle=msg.angle_increment
  LIDARangle=msg.angle_max
  for i in msg.ranges:
    if LIDARangle>0:
      LeftLIDARlist.append(i)
    else:
      RightLIDARlist.insert(0,i)
    LIDARangle-=msg.angle_increment
def SimTime(msg):
  global currentTime
  currentTime = msg.clock.secs

def newOdon(msg):
  global x_new
  global y_new
  x_new=msg.pose.pose.position.x
  y_new=msg.pose.pose.position.y


#subscriptuions
sub = rospy.Subscriber("/odom", Odometry,newOdon)
subImu = rospy.Subscriber("/imu", Imu,procImu)
subLeftIR = rospy.Subscriber("/range/fl", Range,leftRange)
subRightIR = rospy.Subscriber("/range/fr", Range,rightRange)
subLIDAR = rospy.Subscriber("/scan", LaserScan, LIDARprocess)
subClock=rospy.Subscriber("/clock",Clock,SimTime)

#publishing
pubMove = rospy.Publisher("/cmd_vel",Twist,queue_size=2)

rospy.init_node("single_project_node")

#"objectDetection"
#IR
def ObjectDetection(LIR,RIR):
  #determines if an object is ahead detected by IR
  if LIR<IRthreshold or RIR<IRthreshold:
    return True
  else:
    return False
#LIDAR
def LIDARdetection(List):
  #determines if an object is ahead detected by LIDAR
  angleOfLIDAR=0
  global LIDARthreshold
  maxRangeLIDAR=LIDARthreshold
  global incrimentAngle
  cWidthMin=0.2
  for j, i in enumerate(List):
    angleOfLIDAR=incrimentAngle*j
    if angleOfLIDAR <3.1415/4: #max angle considered form straigh ahead
      cWidth=i*math.sin(angleOfLIDAR)
      if i <=maxRangeLIDAR and j in [0,1,2]:
        return True, i
      elif cWidth<cWidthMin and j!=0 and i<=maxRangeLIDAR:
        return True, i
    else:
      return False , maxRangeLIDAR
#testing functions
def DistTravled(xSpeed,ySpeed,RosRate):
    #speed of travel
  time=float(1)/float(RosRate)
  xDist=xSpeed*time
  yDist=ySpeed*time
  Dist=(yDist**2+xDist**2)**0.5
  return Dist
def distTravledOd(x_new,x_old,y_new,y_old):
  x=x_old-x_new
  y=y_old-y_new
  dist=(y**2+x**2)**0.5
  return dist

#defined constants
RosRate=25
normalAngleAtOrigne=math.pi
LIDARthreshold=1.0  #lidar threshold
IRthreshold=0.5     #IR threshold
IRx=0.0             #IR speed of travel
IRz=0.5             #IR angular velocity
LIDARx=0.0          #LIDAR speed of travel
LIDARz=0.5          #LIDAR angular velocity
Retx=0.1            #speed to travel when returning to objective
Retz=0.25           #angular velocity when returning to objective
NormVelx=0.2        #speed when traveling in direction of objective
#program
x_old=x_new
y_old=y_new
speed=Twist()

goal=Point()
flag=0
Count=2
rate = rospy.Rate(RosRate)
RateCount=0
LIDARflag=0
distTravledTotal=0.0
startTime=currentTime
distTravledOdom=0.0

while not rospy.is_shutdown():
  LeftlistLIDARBool,LeftlistLIDARRange=LIDARdetection(LeftLIDARlist)
  RightlistLIDARBool,RightlistLIDARRange=LIDARdetection(RightLIDARlist)
  if Count!=0: #alowes for all initial parameters to be filled
    Count-=1
    speed.linear.x=0.0
    speed.angular.z=0.0

  else:
    if ObjectDetection(Lrange,Rrange):  #IR detecton
```

```
      if Rrange<=IRthreshold and Lrange>=Rrange: #left or right condition
        if flag==1:
          speed.angular.z=-IRz
        else:
          speed.angular.z=IRz
          flag=-1
        speed.linear.x=IRx

      elif Lrange<IRthreshold and Rrange>=Lrange:  #left or right condition
        if flag==-1:
          speed.angular.z=IRz
        else:
          speed.angular.z=-IRz
          flag=1
        speed.linear.x=IRx


      else:
        speed.linear.x=0.05
        speed.angular.z=0.0
      RateCount=RosRate*4
      LIDARflag=0
    elif LeftlistLIDARBool: #left LIDAR detection condition
      if LIDARflag==-1 or (LeftlistLIDARRange>RightlistLIDARRange and LIDARflag!=1):
        speed.angular.z=LIDARz
      else:
        speed.angular.z=-LIDARz
        LIDARflag=1
      speed.linear.x=LIDARx
    elif RightlistLIDARBool: #right LIDAR detection condition
      if LIDARflag==1 or (LeftlistLIDARRange<RightlistLIDARRange and LIDARflag!=-1):
        speed.angular.z=-LIDARz
      else:
        speed.angular.z=LIDARz
        LIDARflag=-1
      speed.linear.x=LIDARx

    else:
      #print ’object no longer detected’
      #print angleToNormal
      #gleToNormal<3.14159 and angleToNormal>0
      if RateCount!=0:  #IR straight forward
        speed.linear.x=0.2
        speed.angular.z=0.0
        RateCount-=1
      elif angleToNormal>0.05:    #return to direction of objective
        #speed.linear.x=0.2
        speed.linear.x=Retx
        speed.angular.z=-Retz
        #print ’turning Left’
      elif angleToNormal<-0.05:       #return to direction of objective
        #speed.linear.x=0.2
        speed.linear.x=Retx
        speed.angular.z=Retz
      else:                       #travel straight ahead
        speed.linear.x=NormVelx
        speed.angular.z=0.0

      LIDARflag=0
      flag=0
  distTravledTotal+=DistTravled(speed.linear.x,0,RosRate)
  distTravledOdom+=distTravledOd(x_new,x_old,y_new,y_old)
  print 'time: ',currentTime-startTime,' dist: ', distTravledTotal, ' odom:', distTravledOdom
  x_old=x_new
  y_old=y_new
  pubMove.publish(speed)
  rate.sleep()
```