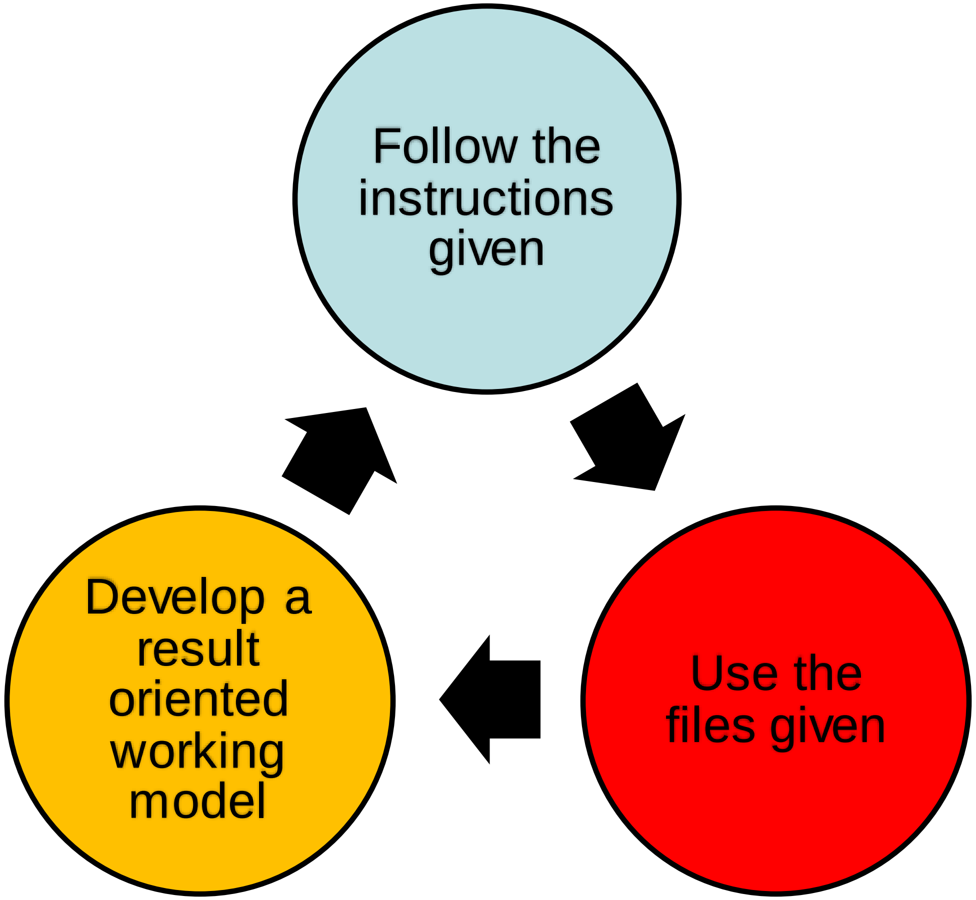
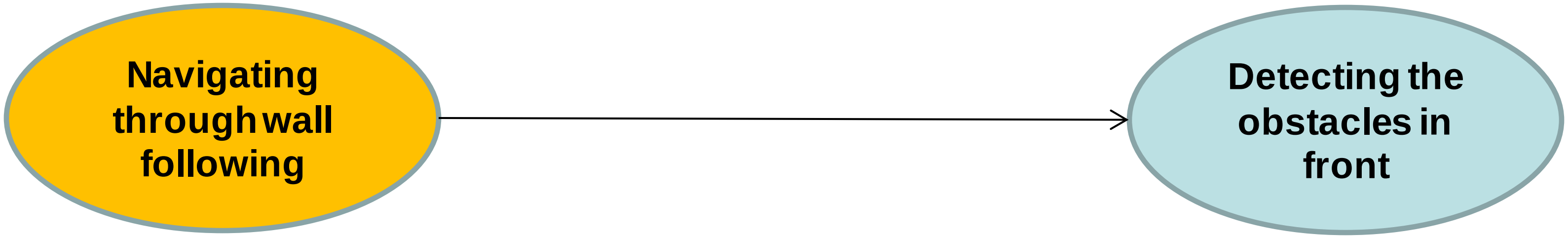
ACTION PLAN:


Follow the instructions given
Use the files given
Develop a result oriented working model

Wall following and Obstacle Avoidance:

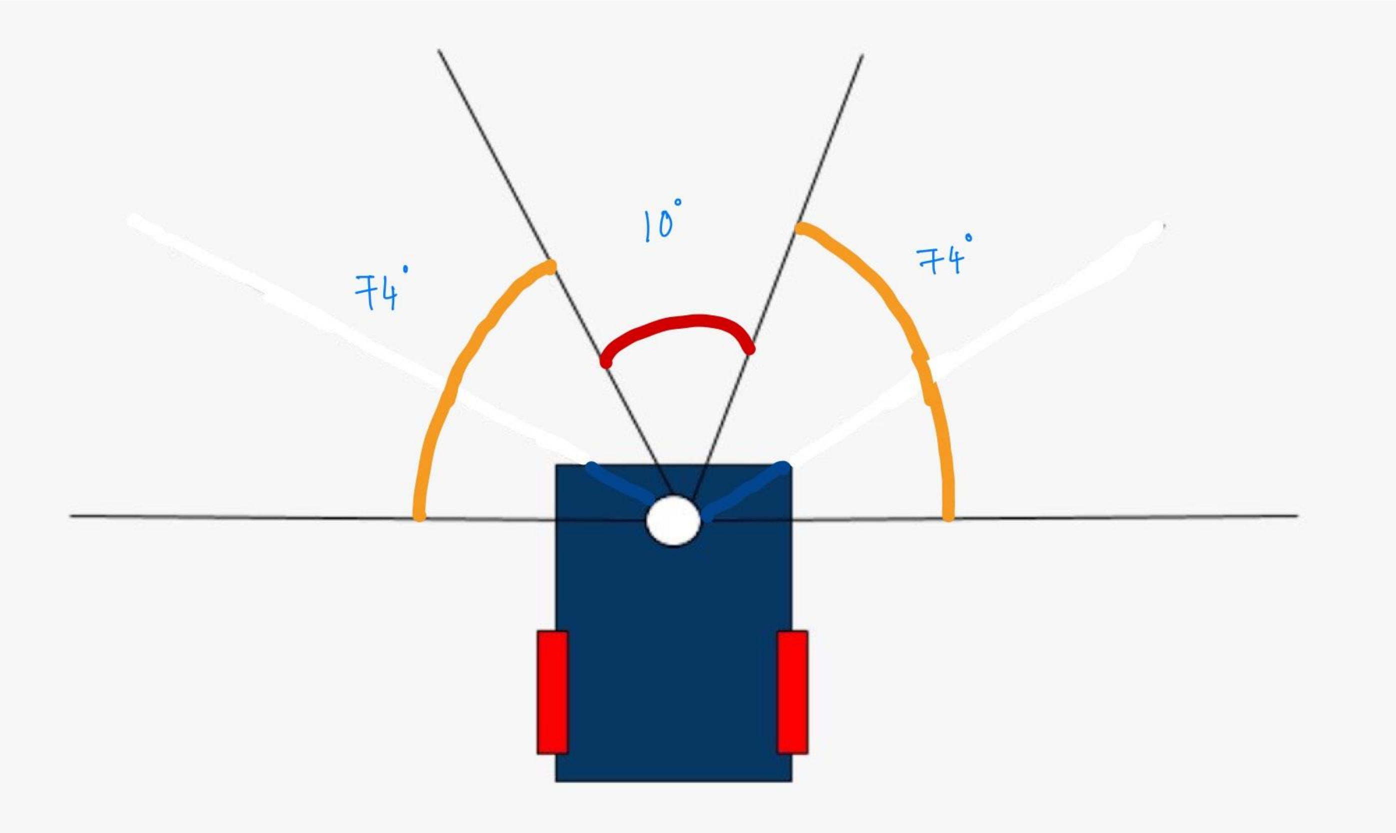
- TurtleBot should be successfully able to follow the wall and avoid the obstacle until it reaches the yellow line – In both Simulation (Gazebo) and Real-world.



Navigating through wall following

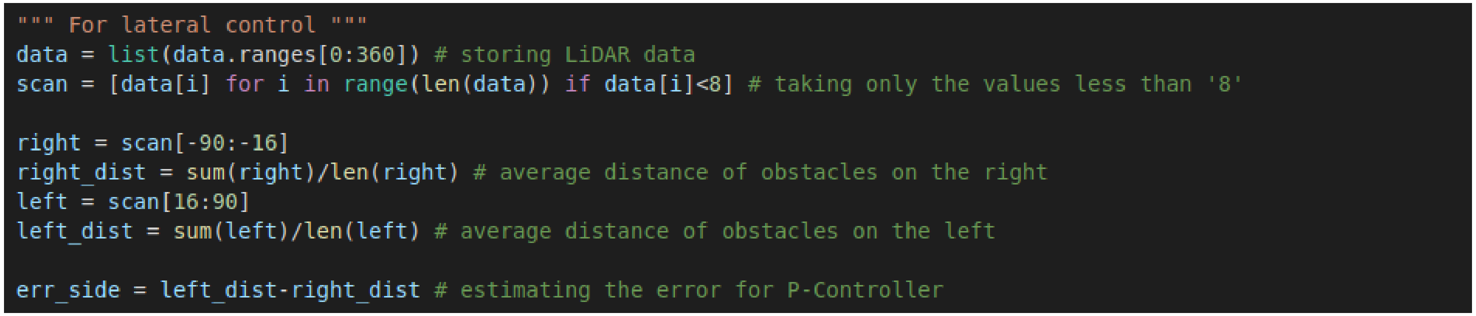
Detecting the obstacles in front

Sector Distribution for Laser Scan



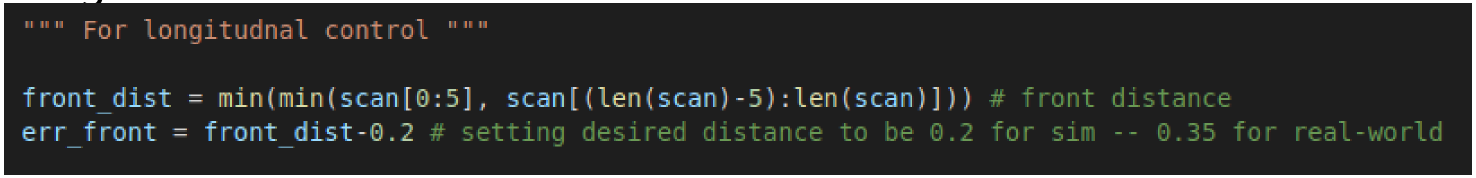
Code/Logic

Lateral Control -



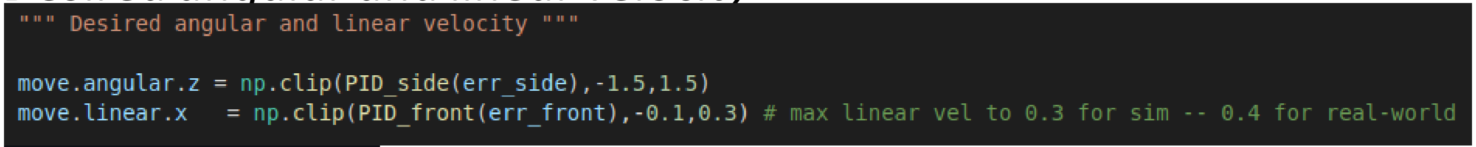
```
""" For lateral control """
data = list(data.ranges[0:360]) # storing LiDAR data
scan = [data[i] for i in range(len(data)) if data[i]<8] # taking only the values less than '8'

right = scan[-90:-16]
right_dist = sum(right)/len(right) # average distance of obstacles on the right
left = scan[16:90]
left_dist = sum(left)/len(left) # average distance of obstacles on the left

err_side = left_dist-right_dist # estimating the error for P-Controller
```

Longitudinal Control -

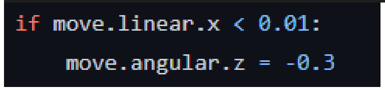


```
""" For longitudnal control """

front_dist = min(min(scan[0:5], scan[(len(scan)-5):len(scan)])) # front distance
err_front = front_dist-0.2 # setting desired distance to be 0.2 for sim -- 0.35 for real-world
```

Desired angular and linear velocity -



```
""" Desired angular and linear velocity """

move.angular.z = np.clip(PID_side(err_side),-1.5,1.5)
move.linear.x   = np.clip(PID_front(err_front),-0.1,0.3) # max linear vel to 0.3 for sim -- 0.4 for real-world
```



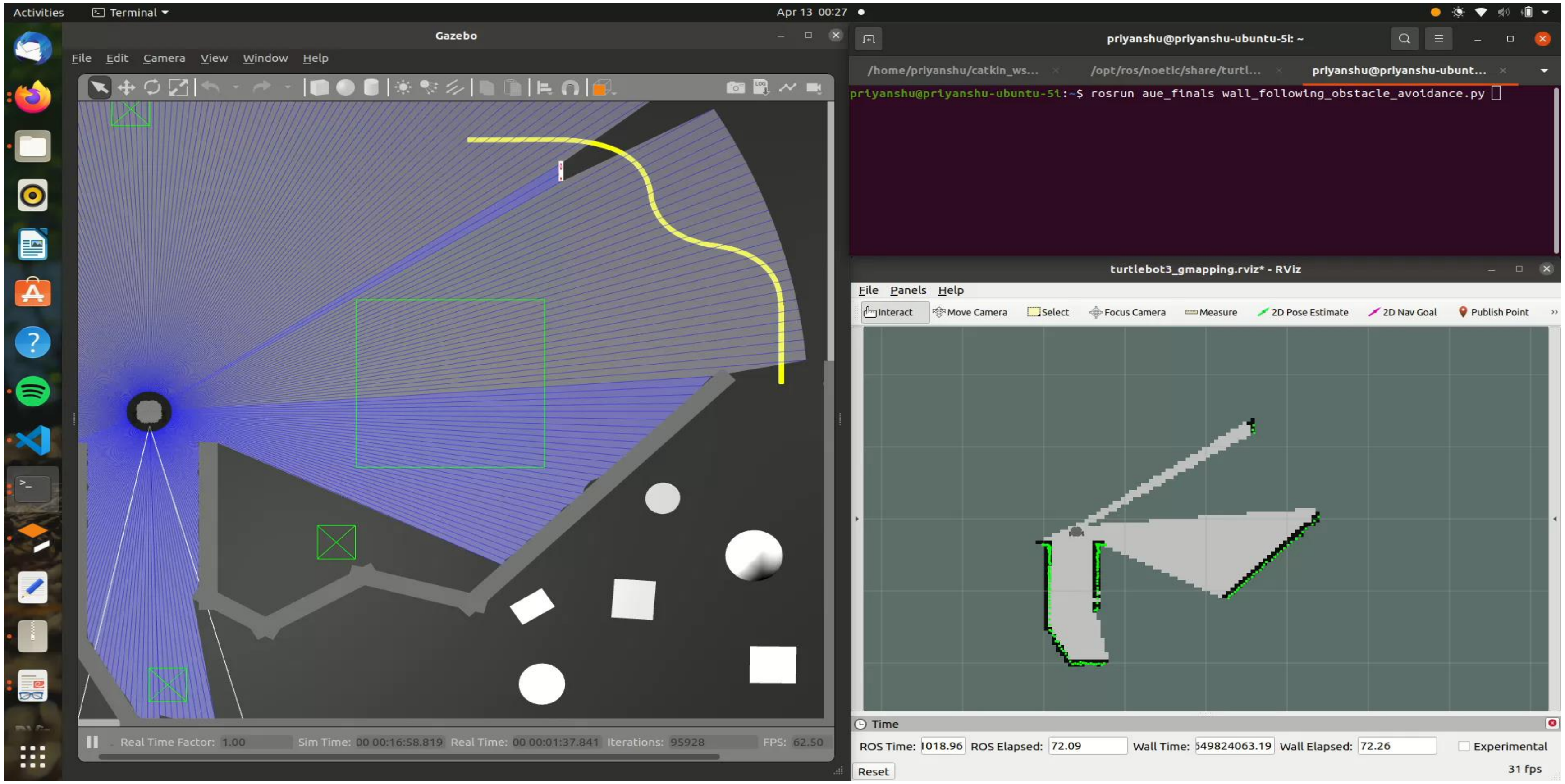
```
if move.linear.x < 0.01:
    move.angular.z = -0.3
```

Code/Logic

Infinity in gazebo is 'inf'

Infinity in real world is '0'

DEMONSTRATION: Simulation (Gazebo)



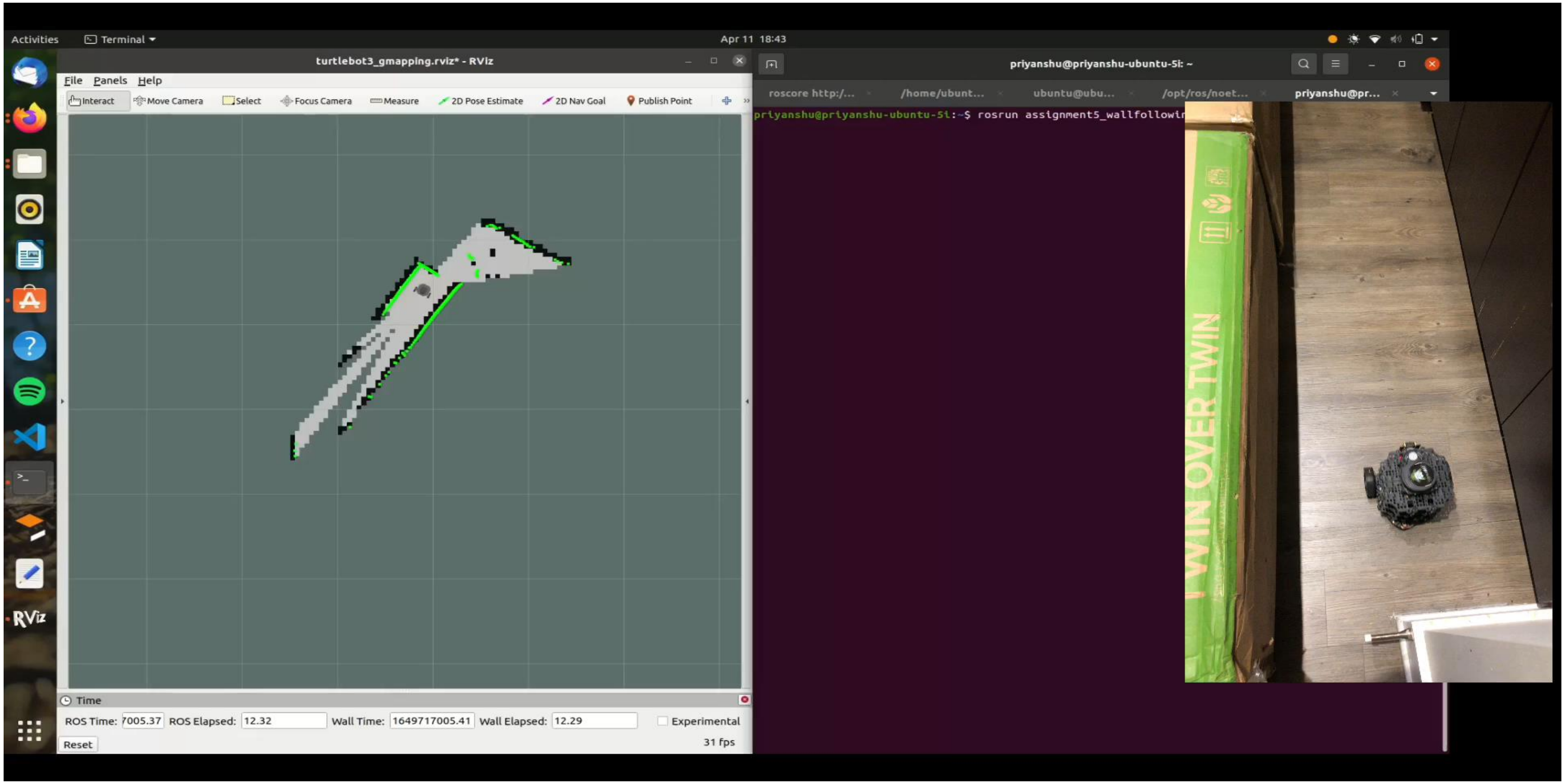
```
priyanshu@priyanshu-ubuntu-5i:~$ rosrun aue_finals wall_following_obstacle_avoidance.py
```

Real Time Factor: 1.00 Sim Time: 00 00:16:58.819 Real Time: 00 00:01:37.841 Iterations: 95928 FPS: 62.50

ROS Time: 1018.96 ROS Elapsed: 72.09 Wall Time: 549824063.19 Wall Elapsed: 72.26 □ Experimental

31 fps

DEMONSTRATION: Real-World



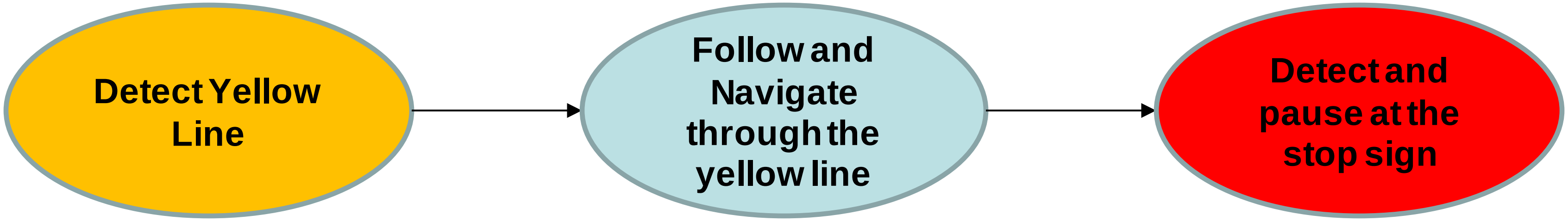
```
priyanshu@priyanshu-ubuntu-5i:~$ rosrun assignment5_wallfollowin
```

ROS Time: 7005.37 ROS Elapsed: 12.32 Wall Time: 1649717005.41 Wall Elapsed: 12.29 □ Experimental

31 fps

Line following and Stop Sign Detection:

- For the task of line following, we use blob tracking.
- We specified a range for the color as lower yellow and upper yellow denoting the range of the colors.
- This method also used masking to detect the blob on the specified color range, increasing the volume of data transfer between the remote PC and the bot.
- Thus, using the compatible input image was an important condition.

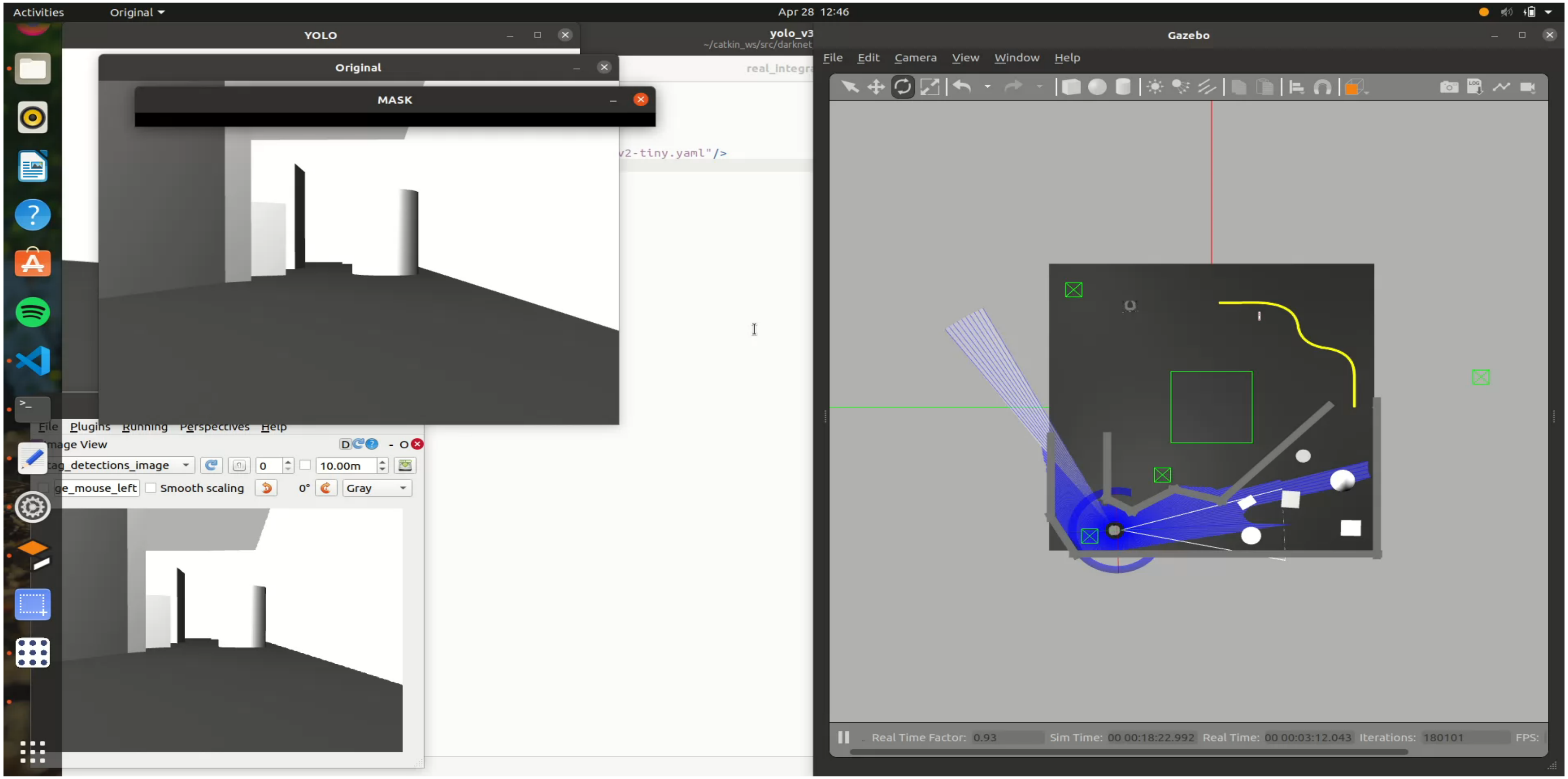


Detect Yellow Line

Follow and Navigate through the yellow line

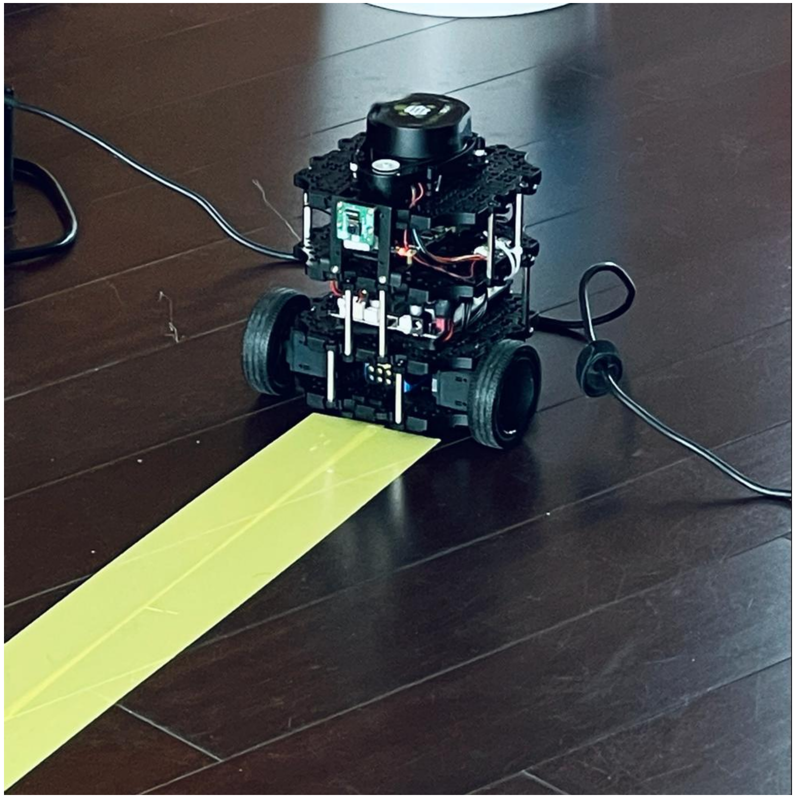
Detect and pause at the stop sign

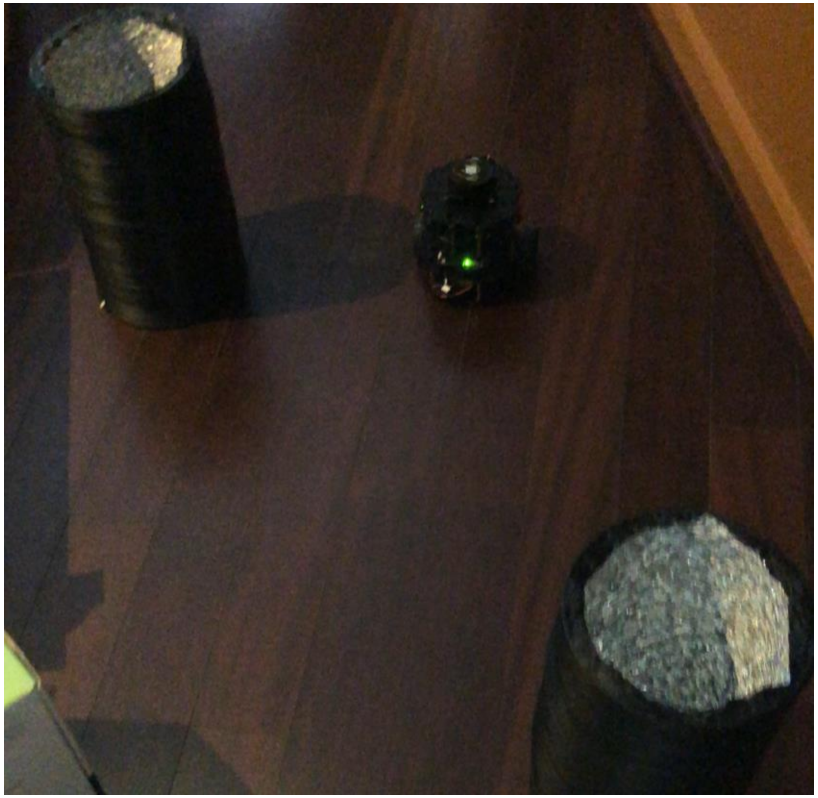
Line Following:



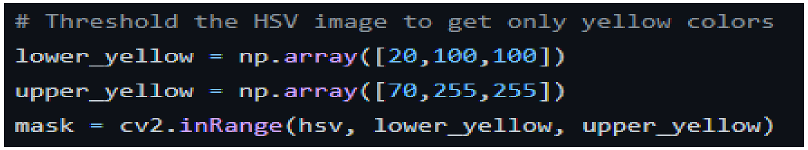
Line Following:

While Testing:

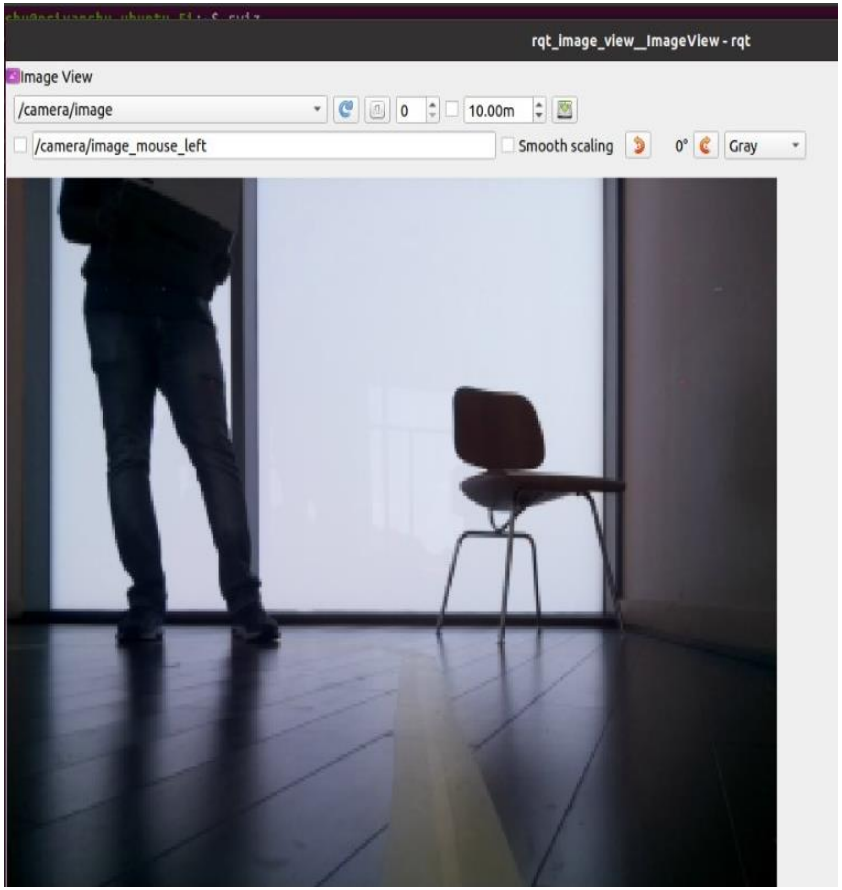




Line Following:



```
# Threshold the HSV image to get only yellow colors
lower_yellow = np.array([20,100,100])
upper_yellow = np.array([70,255,255])
mask = cv2.inRange(hsv, lower_yellow, upper_yellow)
```



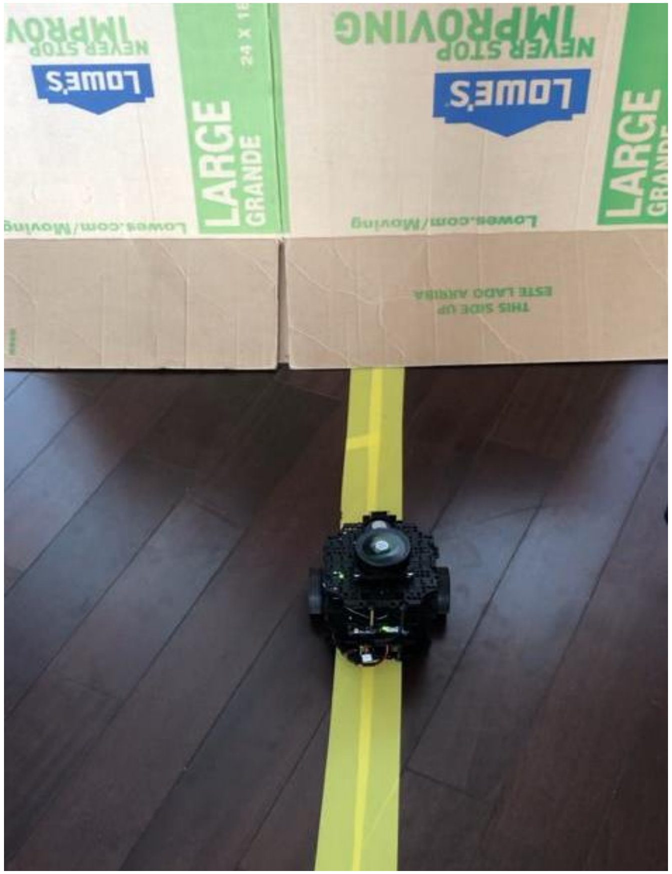
rqt_image_view__ImageView - rqt

Image View

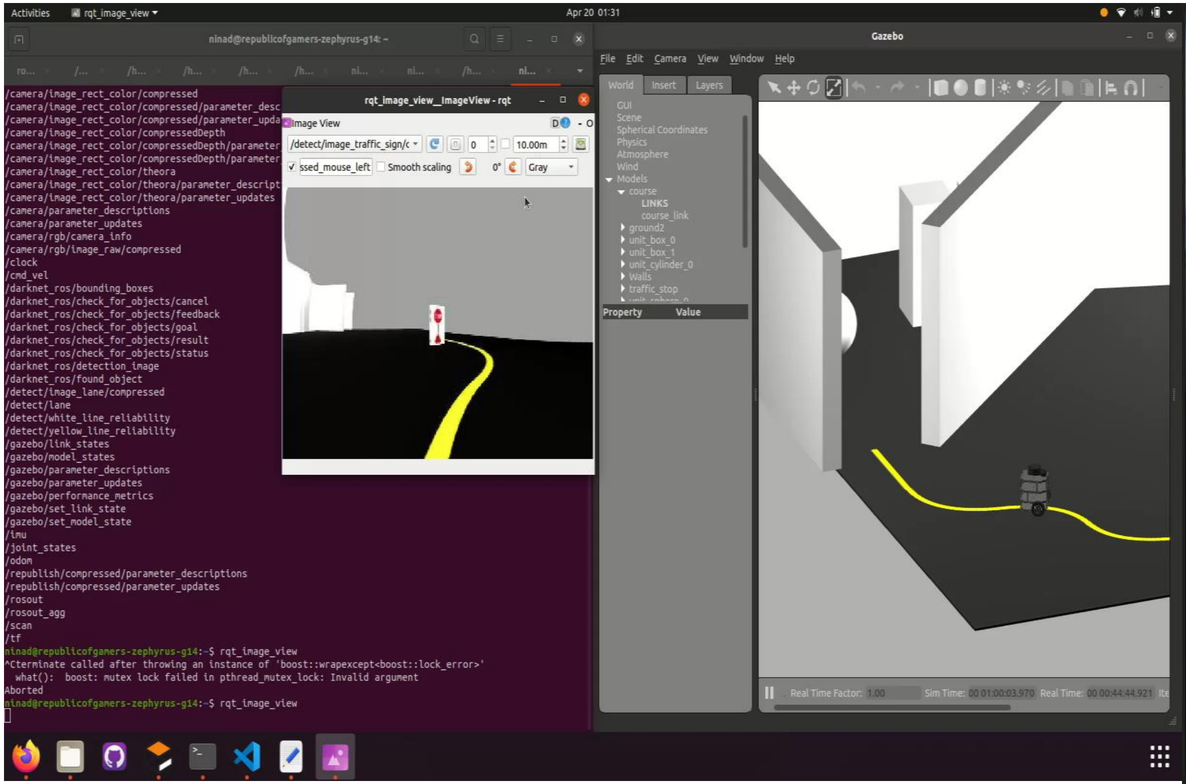
/camera/image 0 ☐ 10.00m

☐ /camera/image_mouse_left ☐ Smooth scaling 0° Gray

To reduce the glare:



Alternate Approaches: (SIFT)

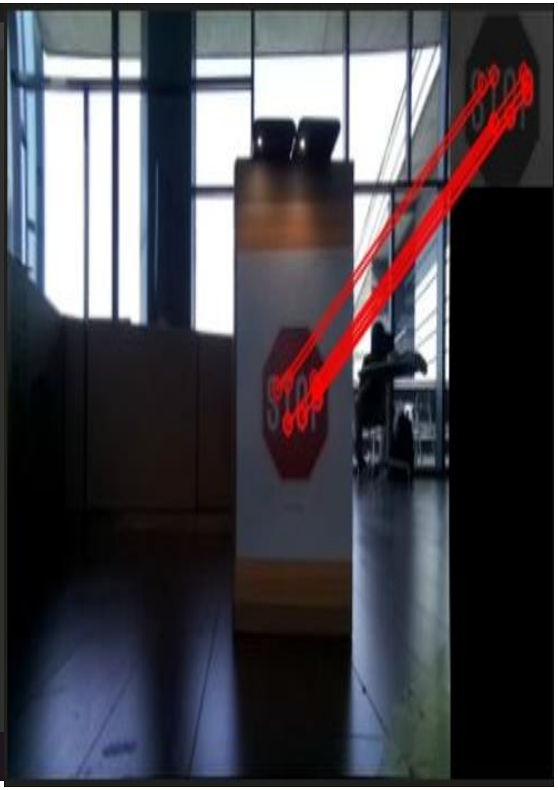


```
/camera/image_rect_color/compressed
/camera/image_rect_color/compressed/parameter_desc
/camera/image_rect_color/compressed/parameter_upda
/camera/image_rect_color/compressedDepth
/camera/image_rect_color/compressedDepth/parameter
/camera/image_rect_color/compressedDepth/parameter
/camera/image_rect_color/theora
/camera/image_rect_color/theora/parameter_descript
/camera/image_rect_color/theora/parameter_updates
/camera/parameter_descriptions
/camera/parameter_updates
/camera/rgb/camera_info
/camera/rgb/image_raw/compressed
/clock
/cmd_vel
/darknet_ros/bounding_boxes
/darknet_ros/check_for_objects/cancel
/darknet_ros/check_for_objects/feedback
/darknet_ros/check_for_objects/goal
/darknet_ros/check_for_objects/result
/darknet_ros/check_for_objects/status
/darknet_ros/detection_image
/darknet_ros/found_object
/detect/image_lane/compressed
/detect/lane
/detect/white_line_reliability
/detect/yellow_line_reliability
/gazebo/link_states
/gazebo/model_states
/gazebo/parameter_descriptions
/gazebo/parameter_updates
/gazebo/performance_metrics
/gazebo/set_link_state
/gazebo/set_model_state
/imu
/joint_states
/odom
/republish/compressed/parameter_descriptions
/republish/compressed/parameter_updates
/rosout
/rosout_agg
/scan
/tf
ninad@republicofgamers-zephyrus-g14:~$ rqt_image_view
^Cterminate called after throwing an instance of 'boost::wrapexcept<boost::lock_error>'
  what():  boost: mutex lock failed in pthread_mutex_lock: Invalid argument
Aborted
ninad@republicofgamers-zephyrus-g14:~$ rqt_image_view
```

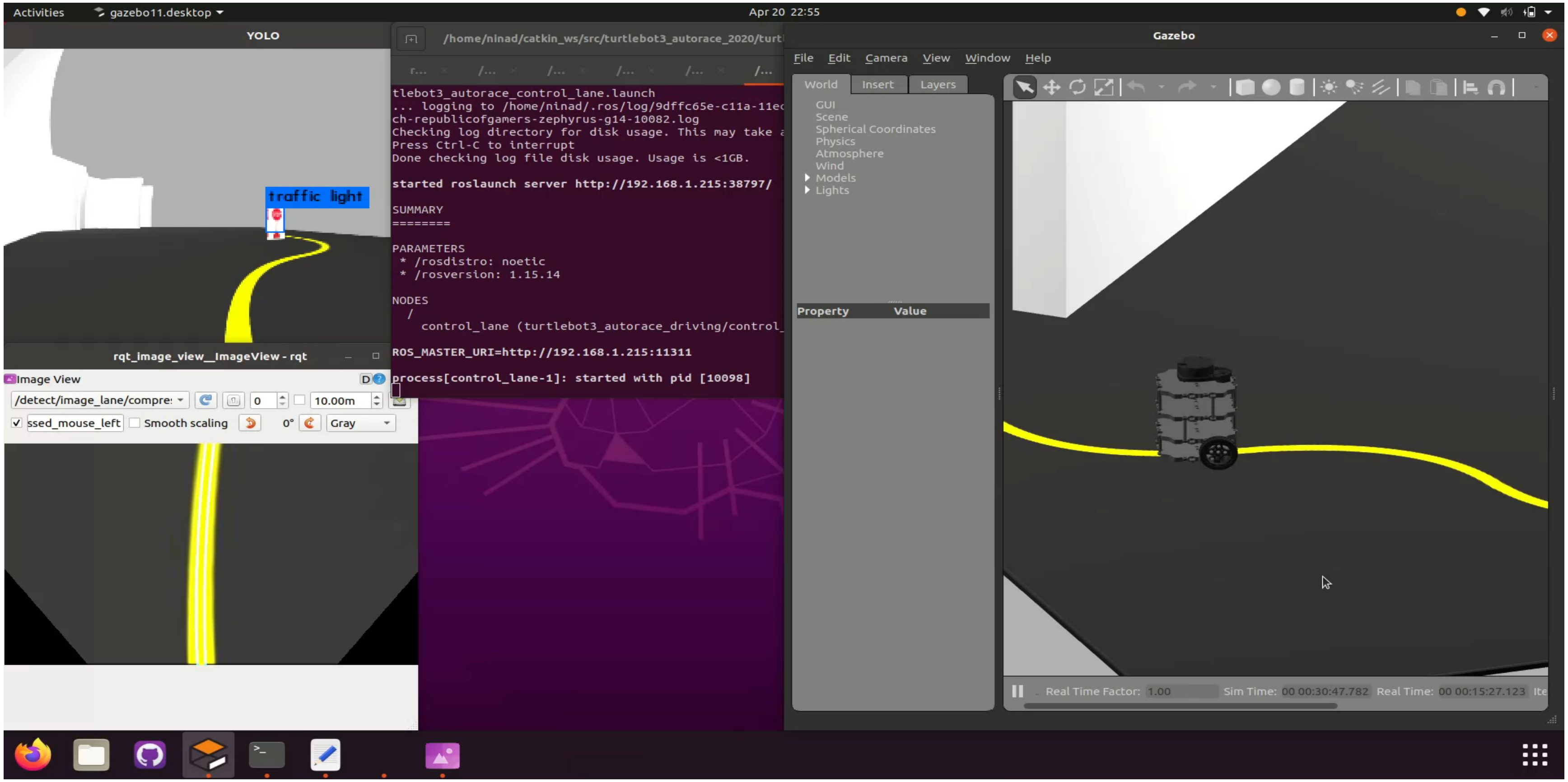
rqt_image_view__ImageView - rqt

Image View: /detect/image_traffic_sign/c, 0, 10.00m, ☑ ssed_mouse_left, □ Smooth scaling, 0°, Gray

Gazebo: Real Time Factor: 1.00 Sim Time: 00 01:00:03.970 Real Time: 00 00:44:44.921



Final Approaches: (TINY YOLO)



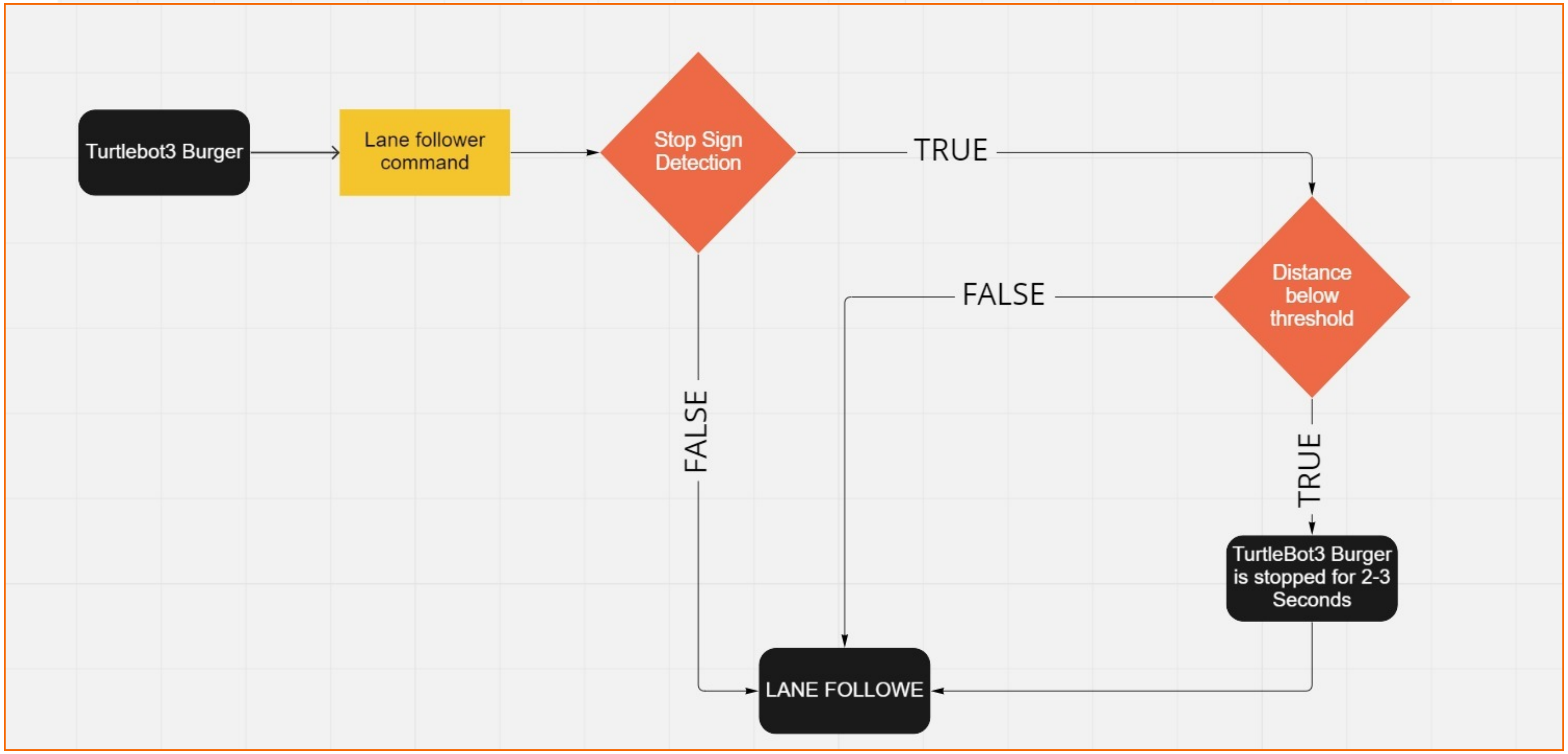
/detect/image_lane/compre: 0 10.00m

☑ ssed_mouse_left □ Smooth scaling 0° Gray

```
tlebot3_autorace_control_lane.launch
... logging to /home/ninad/.ros/log/9dffc65e-c11a-11ec
ch-republicofgamers-zephyrus-g14-10082.log
Checking log directory for disk usage. This may take a
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://192.168.1.215:38797/

SUMMARY
========

PARAMETERS
 * /rosdistro: noetic
 * /rosversion: 1.15.14

NODES
  /
    control_lane (turtlebot3_autorace_driving/control_

ROS_MASTER_URI=http://192.168.1.215:11311

process[control_lane-1]: started with pid [10098]
```

Real Time Factor: 1.00 Sim Time: 00 00:30:47.782 Real Time: 00 00:15:27.123 Ite

Controller:



Turtlebot3 Burger

Lane follower command

Stop Sign Detection

TRUE

Distance below threshold

FALSE

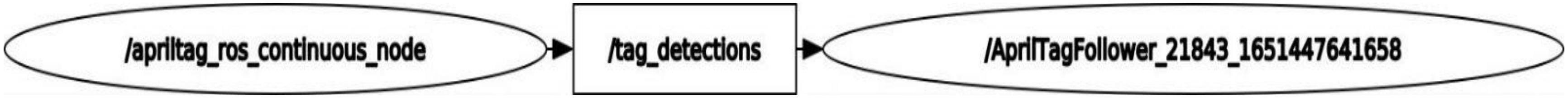
FALSE

TRUE

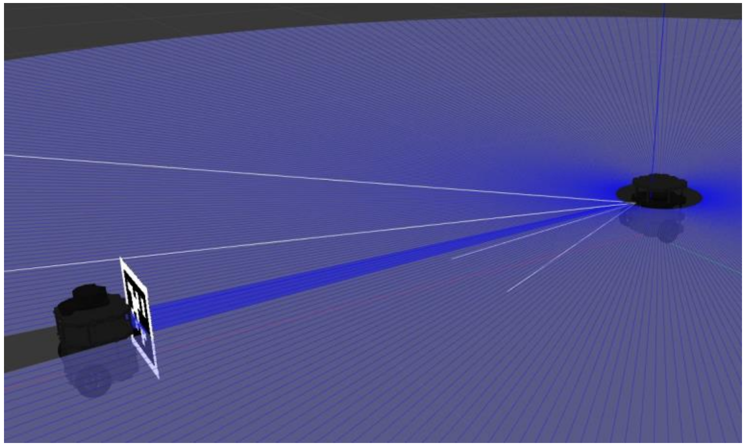
TurtleBot3 Burger is stopped for 2-3 Seconds

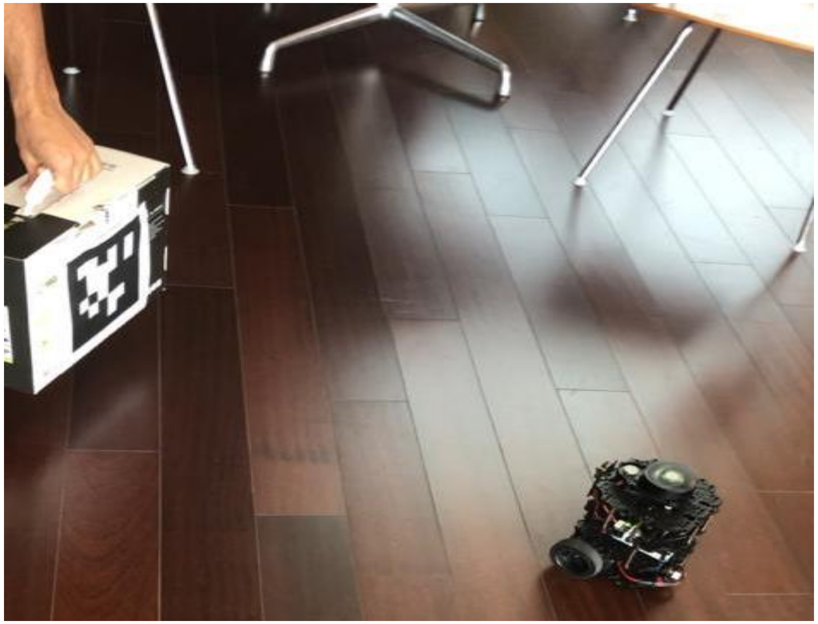
LANE FOLLOWE

April Tag Detection:

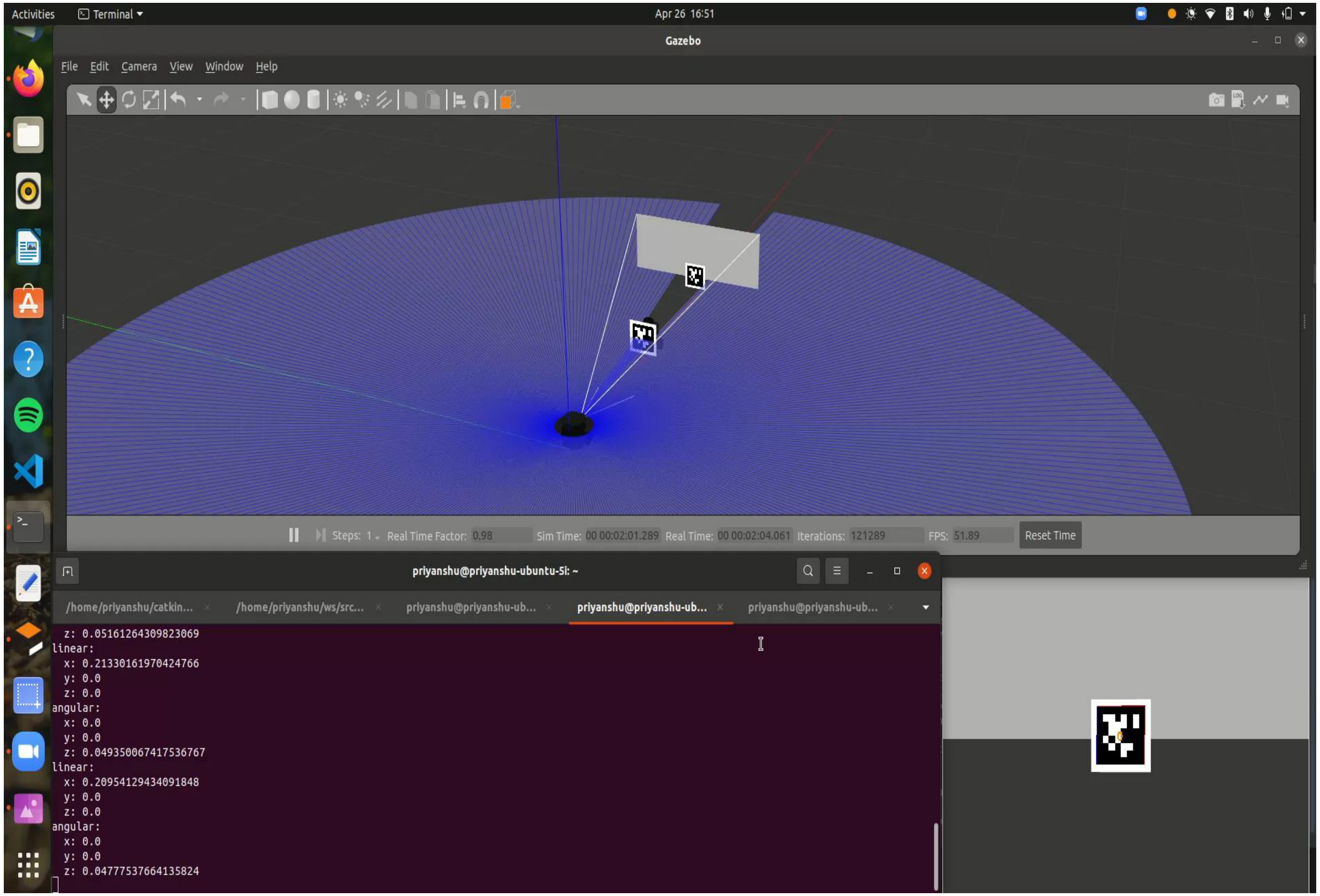


/apriltag_ros_continuous_node → /tag_detections → /AprilTagFollower_21843_1651447641658





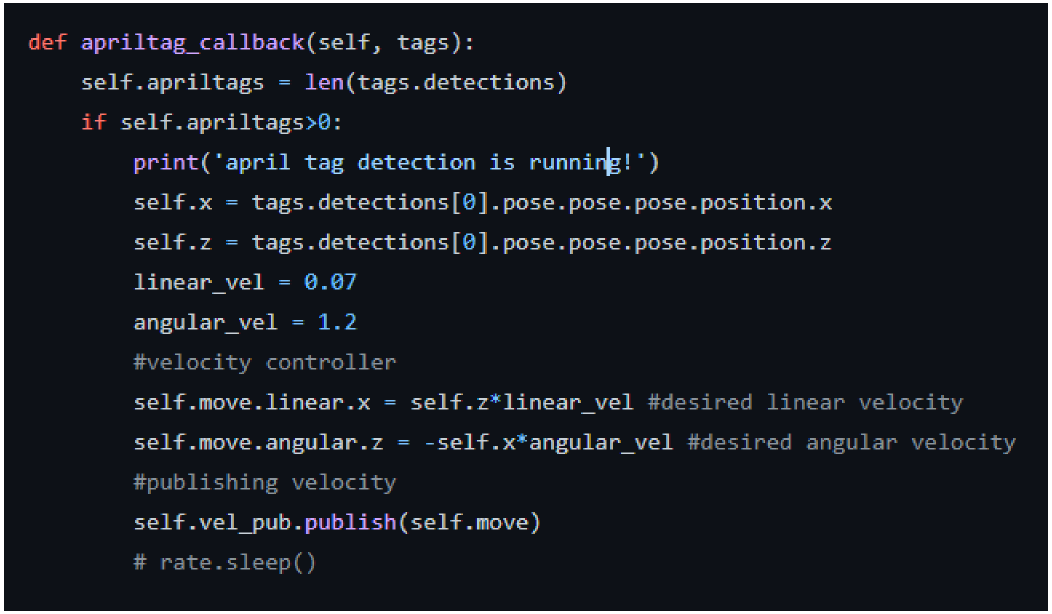
Demonstration: Simulation (Gazebo)


Activities
Terminal
Apr 26 16:51
Gazebo
File
Edit
Camera
View
Window
Help
Steps: 1
Real Time Factor: 0.98
Sim Time: 00 00:02:01.289
Real Time: 00 00:02:04.061
Iterations: 121289
FPS: 51.89
Reset Time
priyanshu@priyanshu-ubuntu-5i: ~
/home/priyanshu/catkin...
/home/priyanshu/ws/src...
priyanshu@priyanshu-ub...
priyanshu@priyanshu-ub...
priyanshu@priyanshu-ub...

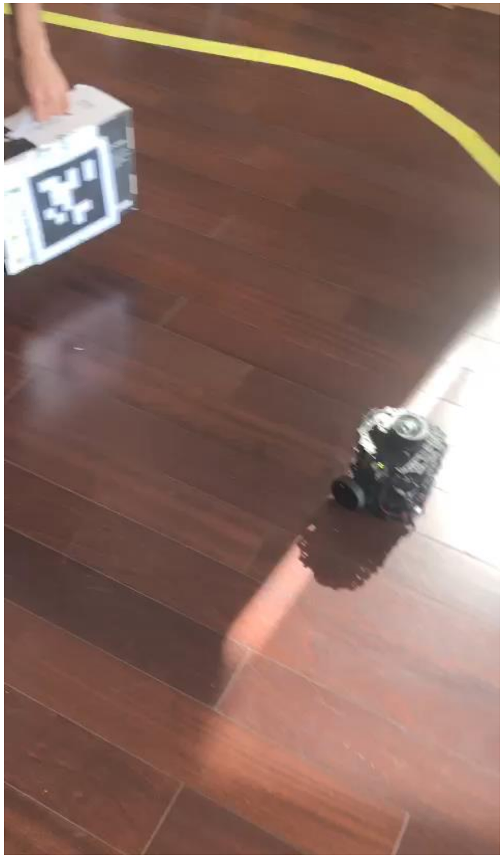
```
  z: 0.05161264309823069
linear:
  x: 0.21330161970424766
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.04935006741753676 7
linear:
  x: 0.20954129434091848
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.04777537664135824
```

Demonstration: Real world

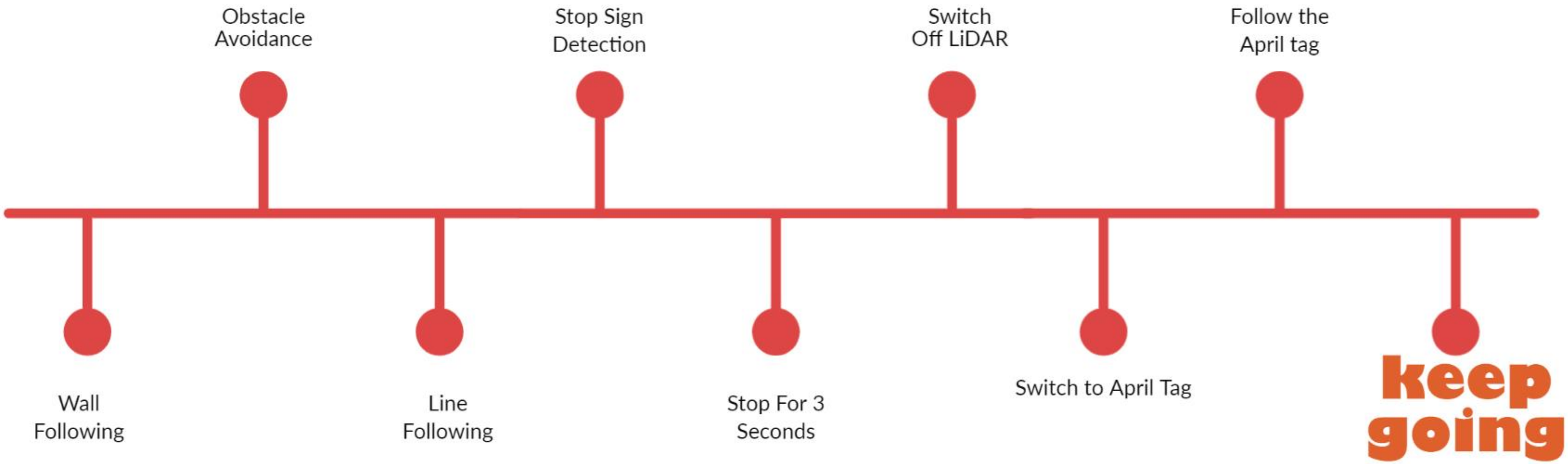
- If ‘tags.detections’ array is not empty
  then follow the ‘April Tag’



```
def apriltag_callback(self, tags):
    self.apriltags = len(tags.detections)
    if self.apriltags>0:
        print('april tag detection is running!')
        self.x = tags.detections[0].pose.pose.pose.position.x
        self.z = tags.detections[0].pose.pose.pose.position.z
        linear_vel = 0.07
        angular_vel = 1.2
        #velocity controller
        self.move.linear.x = self.z*linear_vel #desired linear velocity
        self.move.angular.z = -self.x*angular_vel #desired angular velocity
        #publishing velocity
        self.vel_pub.publish(self.move)
        # rate.sleep()
```



Integration of tasks



Wall Following

Obstacle Avoidance

Line Following

Stop Sign Detection

Stop For 3 Seconds

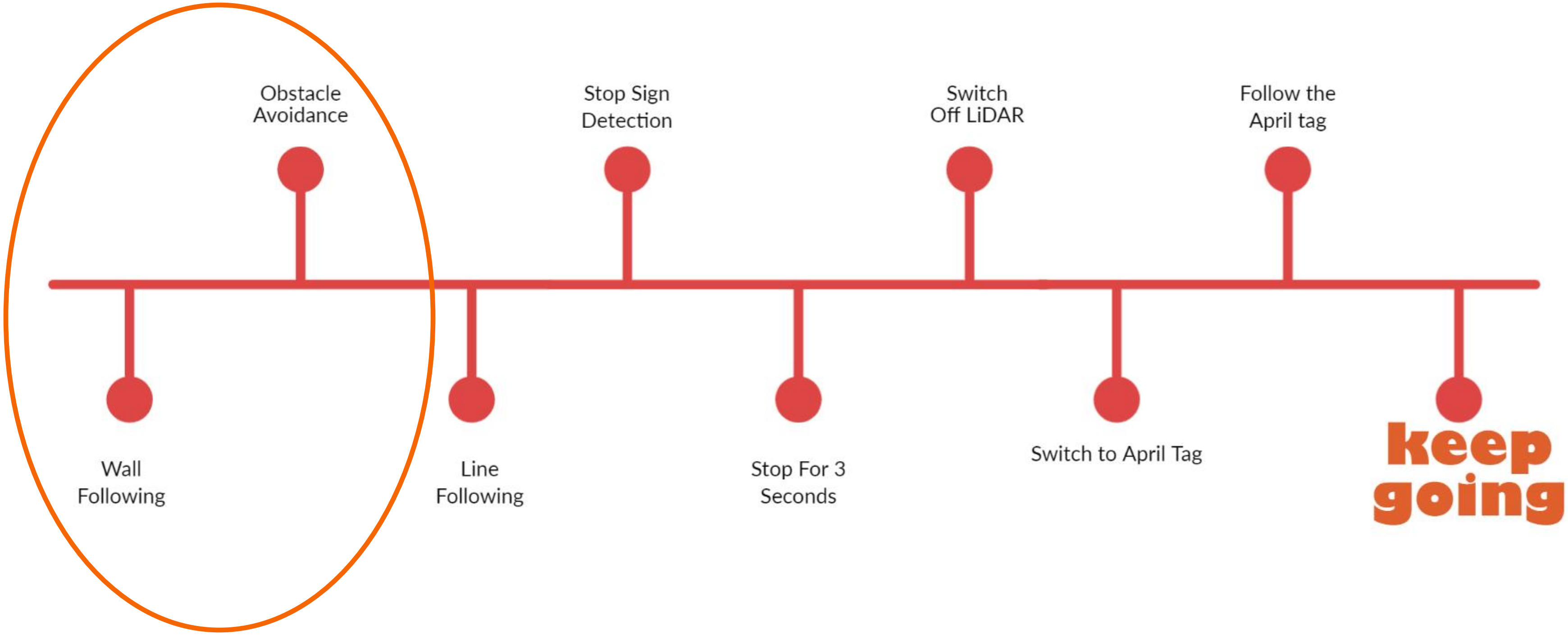
Switch Off LiDAR

Switch to April Tag

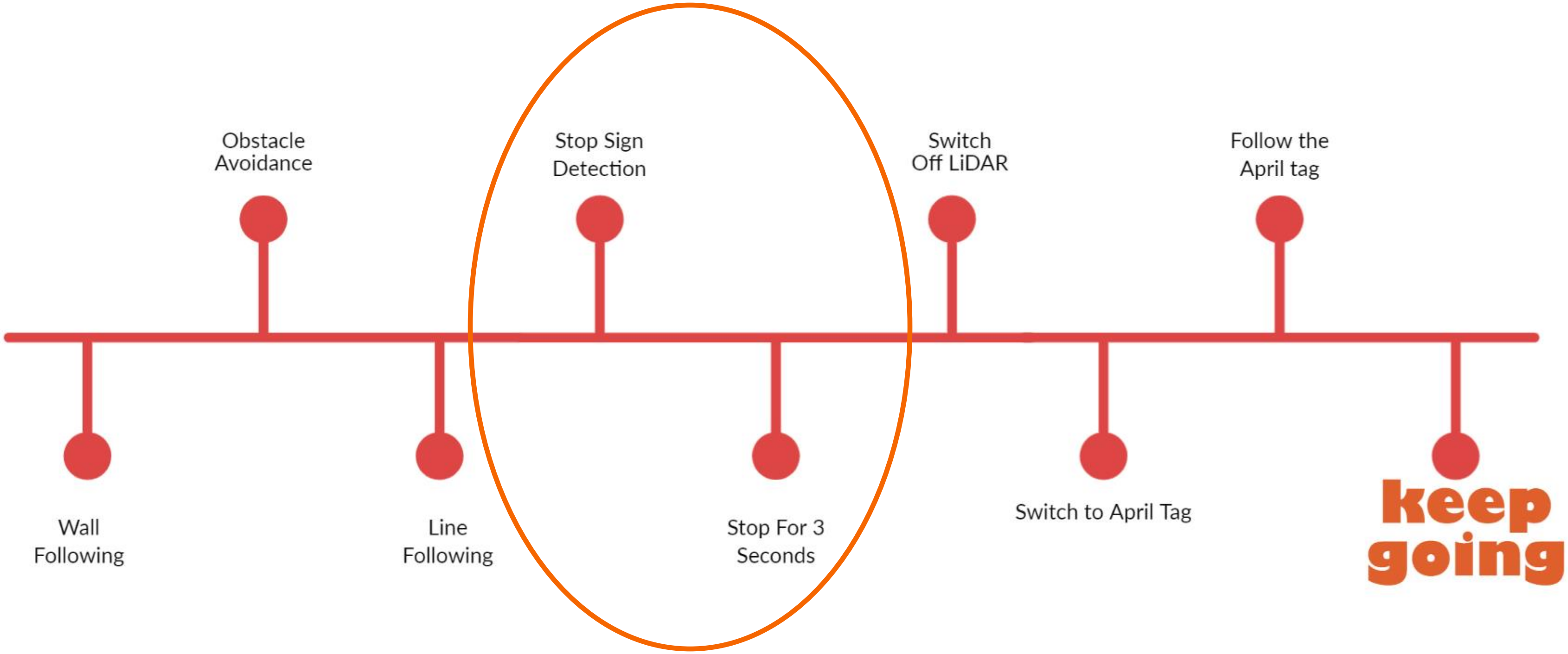
Follow the April tag

keep going

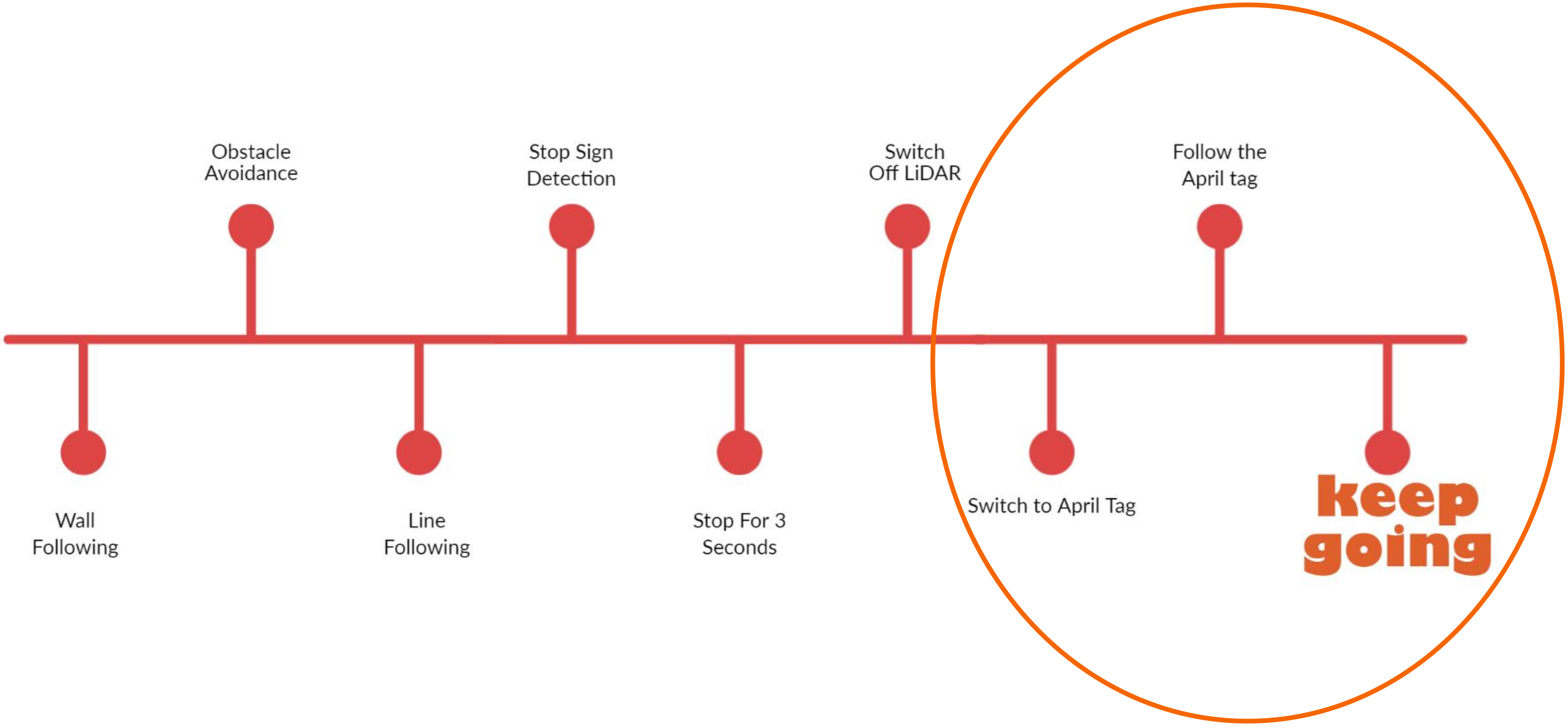
Integration of tasks



Integration of tasks



Integration of tasks



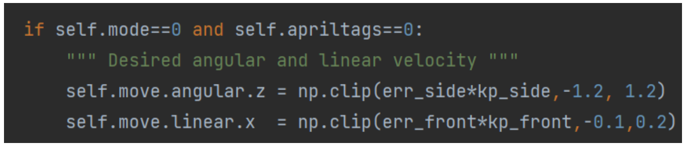
Pseudo Code

```
While (1)
        Line Follower logic
        If (obstacles < front optimal distance)
                 Obstacle avoidance
        Else if ( obstacles < lateral optimal distance)
                 Wall Following
        end
        If ( stop sign = 1)
               Pause for 3 seconds
         end
         If ( April tag = 1)
            April tag detection ()
            break
         end
end
```

Integration of tasks

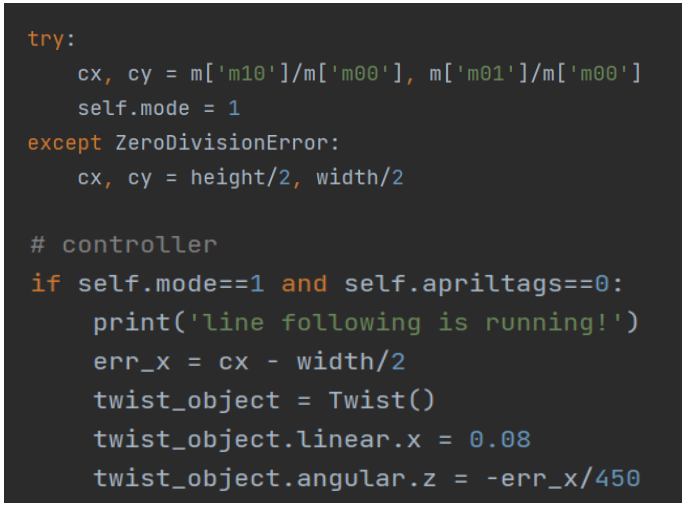
- For integrating all the tasks, we used the line following code as our base code which will keep on running all the time with switch-cases built in it to switch to different tasks.

1. Firstly, we set mode=0 which will run the controller for the wall following and obstacle avoidance.



```
if self.mode==0 and self.apriltags==0:
    """ Desired angular and linear velocity """
    self.move.angular.z = np.clip(err_side*kp_side,-1.2, 1.2)
    self.move.linear.x  = np.clip(err_front*kp_front,-0.1,0.2)
```

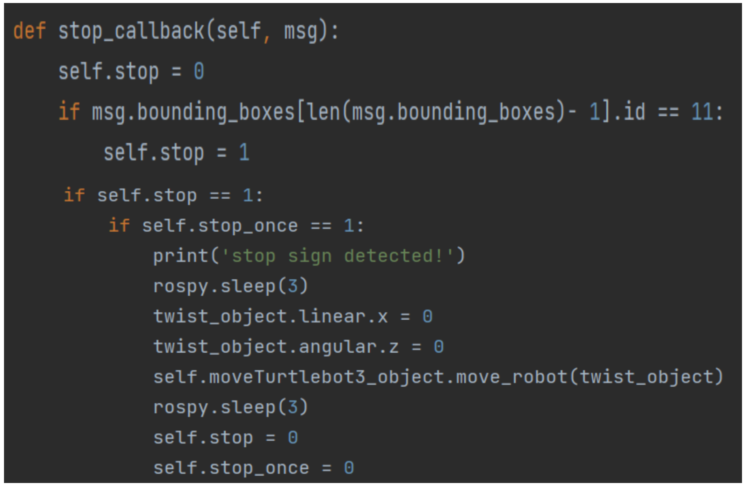
2. Secondly, once the camera detects the yellow line (blob), the line following code finds a centroid for it after which the mode is set to 1, which will change the controller to that defined for line following.



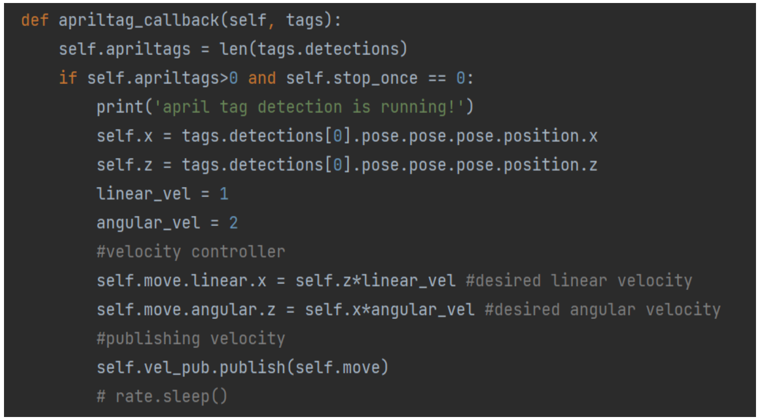
```
try:
    cx, cy = m['m10']/m['m00'], m['m01']/m['m00']
    self.mode = 1
except ZeroDivisionError:
    cx, cy = height/2, width/2

# controller
if self.mode==1 and self.apriltags==0:
    print('line following is running!')
    err_x = cx - width/2
    twist_object = Twist()
    twist_object.linear.x = 0.08
    twist_object.angular.z = -err_x/450
```

Integration of tasks

3. After which a flag for stop sign was built in the code to publish zero linear and angular velocity once the tiny yolo algorithm detects a stop sign. Furthermore, logic was added such that the robot stops only once for 3 seconds.



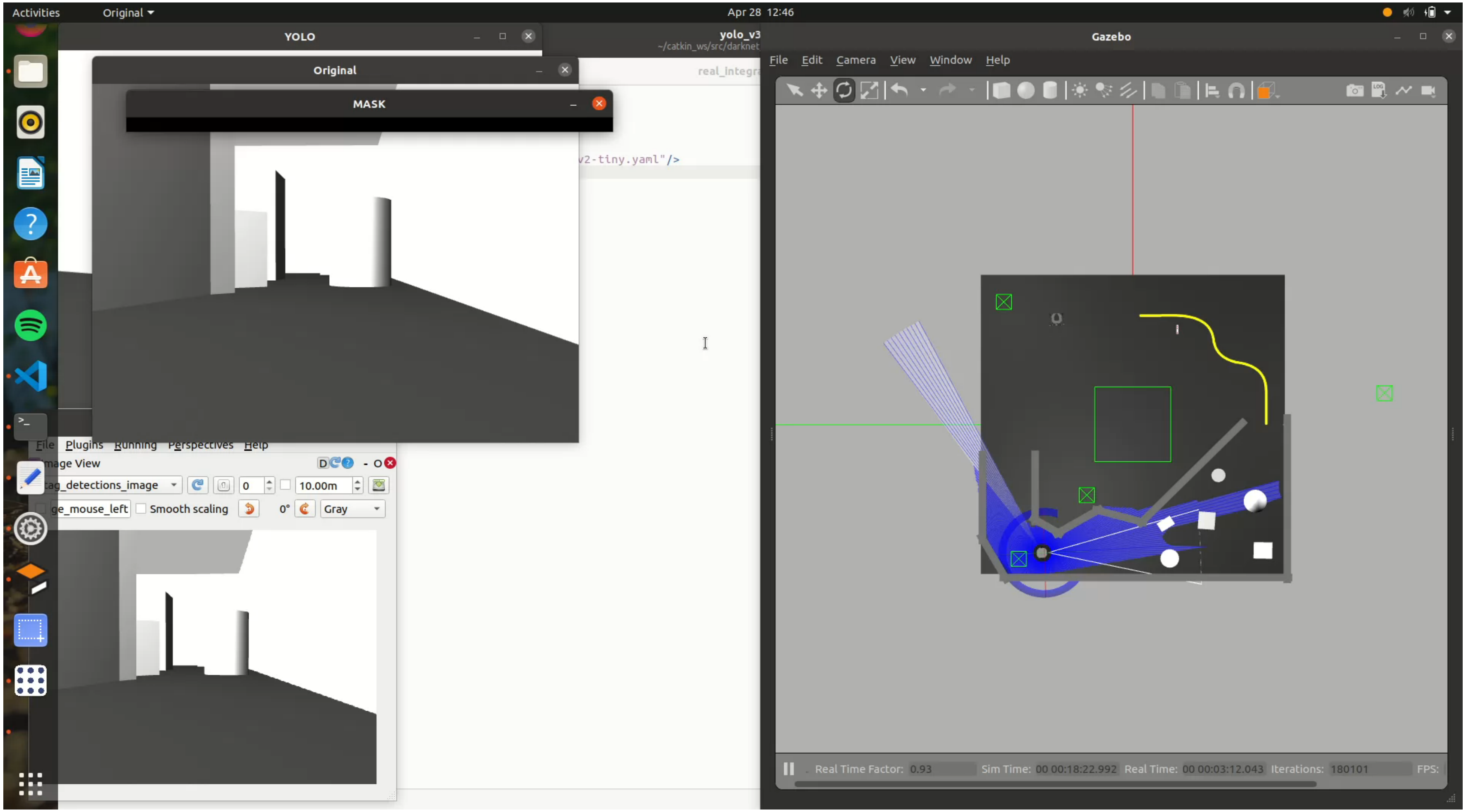
```python
def stop_callback(self, msg):
    self.stop = 0
    if msg.bounding_boxes[len(msg.bounding_boxes)- 1].id == 11:
        self.stop = 1
    if self.stop == 1:
        if self.stop_once == 1:
            print('stop sign detected!')
            rospy.sleep(3)
            twist_object.linear.x = 0
            twist_object.angular.z = 0
            self.moveTurtlebot3_object.move_robot(twist_object)
            rospy.sleep(3)
            self.stop = 0
            self.stop_once = 0
```



```python
def apriltag_callback(self, tags):
    self.apriltags = len(tags.detections)
    if self.apriltags>0 and self.stop_once == 0:
        print('april tag detection is running!')
        self.x = tags.detections[0].pose.pose.pose.position.x
        self.z = tags.detections[0].pose.pose.pose.position.z
        linear_vel = 1
        angular_vel = 2
        #velocity controller
        self.move.linear.x = self.z*linear_vel #desired linear velocity
        self.move.angular.z = self.x*angular_vel #desired angular velocity
        #publishing velocity
        self.vel_pub.publish(self.move)
        # rate.sleep()
```

4. After this, we added a switch case for the April tag detection which prioritize the controller for the April tag following over other controllers within the code.

Demonstration: Simulation (Gazebo)



tag_detections_image 0 10.00m

ge_mouse_left ☐ Smooth scaling 0° Gray

Real Time Factor: 0.93 Sim Time: 00 00:18:22.992 Real Time: 00 00:03:12.043 Iterations: 180101 FPS:

Things We Tried To Make The Integration Seamless!

1. Distributing all the nodes on two computers running on the same network as the TurtleBot.

2. Running some nodes on the TurtleBot like the camera calibration packages.

3. Topic remapping and using compressed image instead of default image.

Challenges

1. Keeping the TurtleBot in constant motion.

2. Round obstacles were difficult to detect when encountered.

3. Tuning the gains for the linear and angular velocity for line following in real world was challenging compared to that in simulation.

4. Tuning the HSV parameters according to the lightning conditions at exact same time of test.

Conclusion

- This course has helped us learn a lot of new concepts in the field of autonomy and implementing them on a TurtleBot3 was an experience of another level.

- The project gave us an understanding of the differences between the simulation and real-world problems that we will encounter when working on similar problems in the future.

- It also gives an idea about how integrating modules can become a tedious and tough task if it is not planned and executed effectively.

Thank you!