Group G1
SES Project Final Presentation

Path Planning for Autonomous Drone

Task Distribution

Mihir Kulkarni - Map Manager and Collision Checking

Srisreyas S - Rapidly Exploring Random Graph Based Planner

Aditya Bidwai - Object Detection Using YOLO

Shivangi Gupta - RRT, RRT*, Physical Integration of Drone

Vishal Singh - RotorS and Flight Controller

ROS Nodes

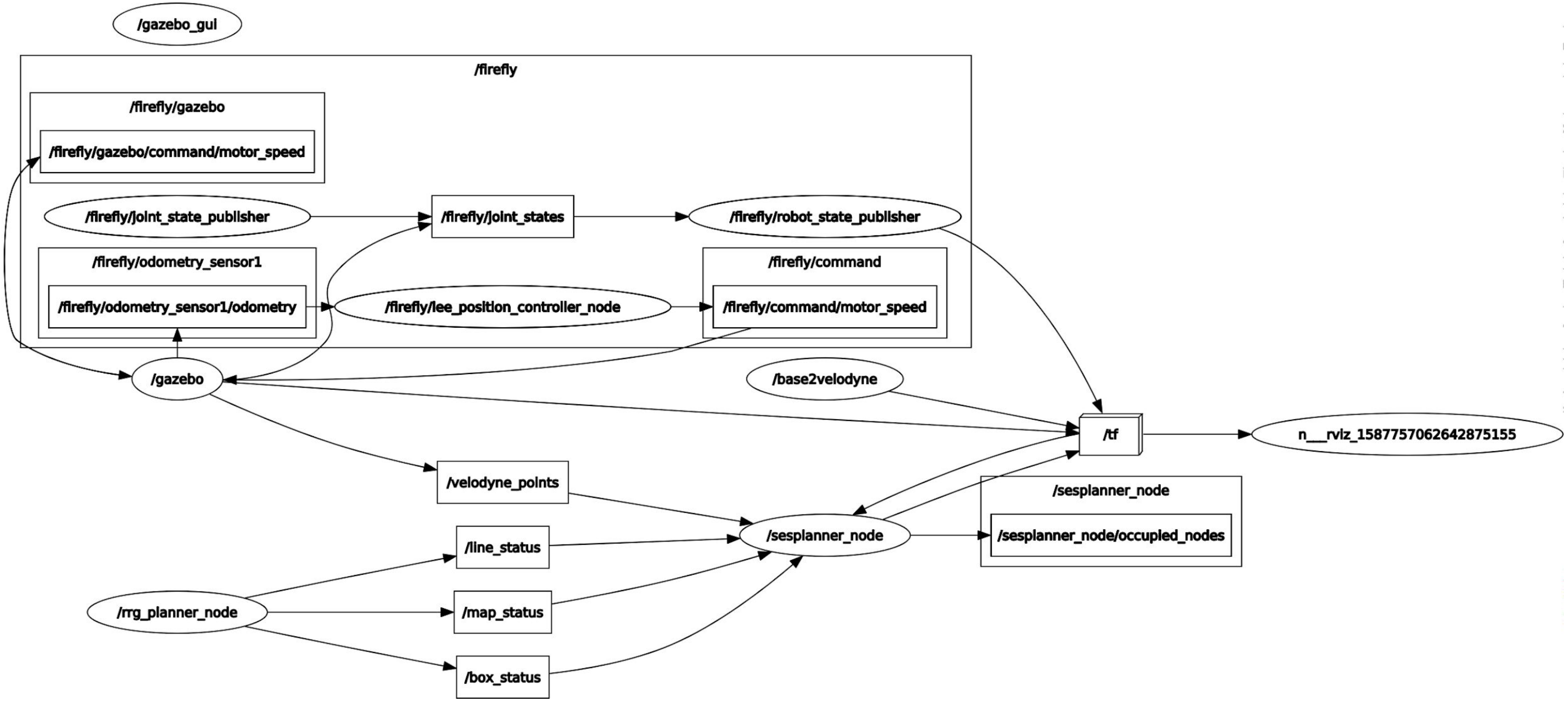
- /gazebo_gui
- /firefly/joint_state_publisher
- /gazebo
- /firefly/lee_position_controller_node
- /firefly/robot_state_publisher
- /sesplanner_node
- /rrgplanner_node
- /base2velodyne
- /camera_red_iris
-

ROS Packages

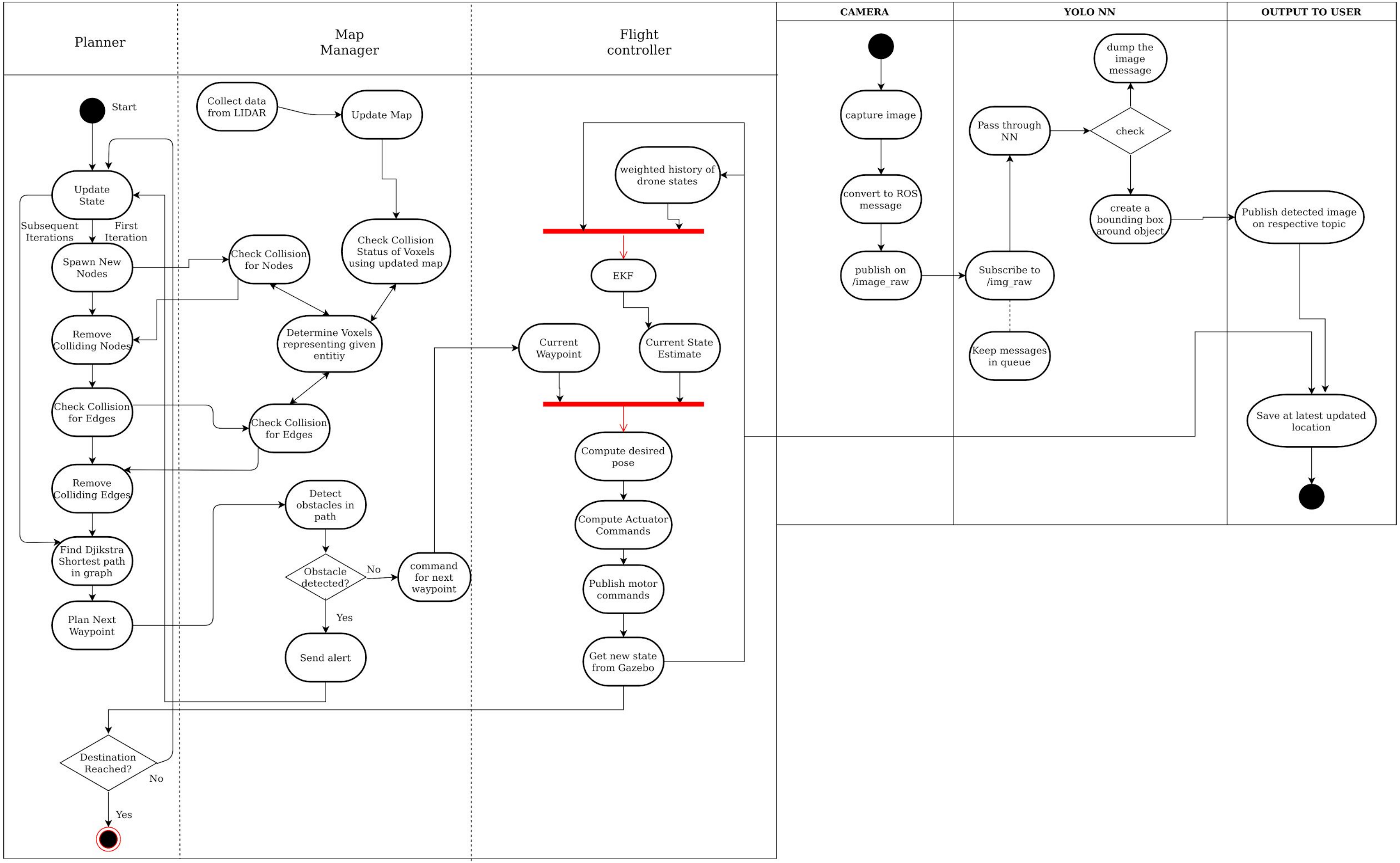
Packages Used

- ses_planner
- voxblox_ros
- velodyne_simulator
- rotors_simulator
- mavlink
- mavros
- eigen_catkin
- catkin_simple
- numpy_eigen
- mavcomm

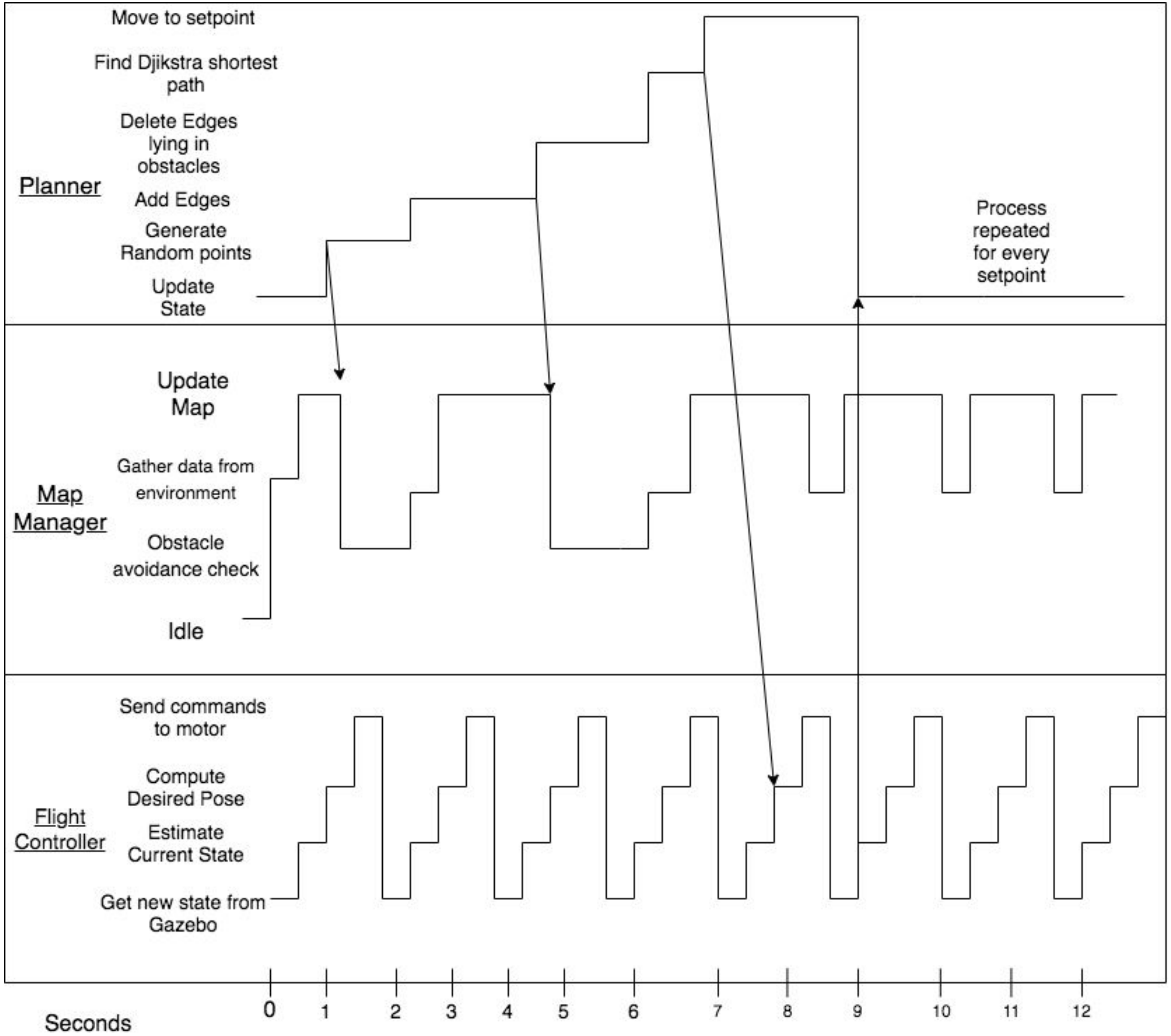
rqt-graph


/gazebo_gui
/firefly
/firefly/gazebo
/firefly/gazebo/command/motor_speed
/firefly/joint_state_publisher
/firefly/joint_states
/firefly/robot_state_publisher
/firefly/odometry_sensor1
/firefly/odometry_sensor1/odometry
/firefly/lee_position_controller_node
/firefly/command
/firefly/command/motor_speed
/gazebo
/base2velodyne
/tf
n__rviz_1587757062642875155
/velodyne_points
/sesplanner_node
/sesplanner_node/occupied_nodes
/line_status
/sesplanner_node
/rrg_planner_node
/map_status
/box_status

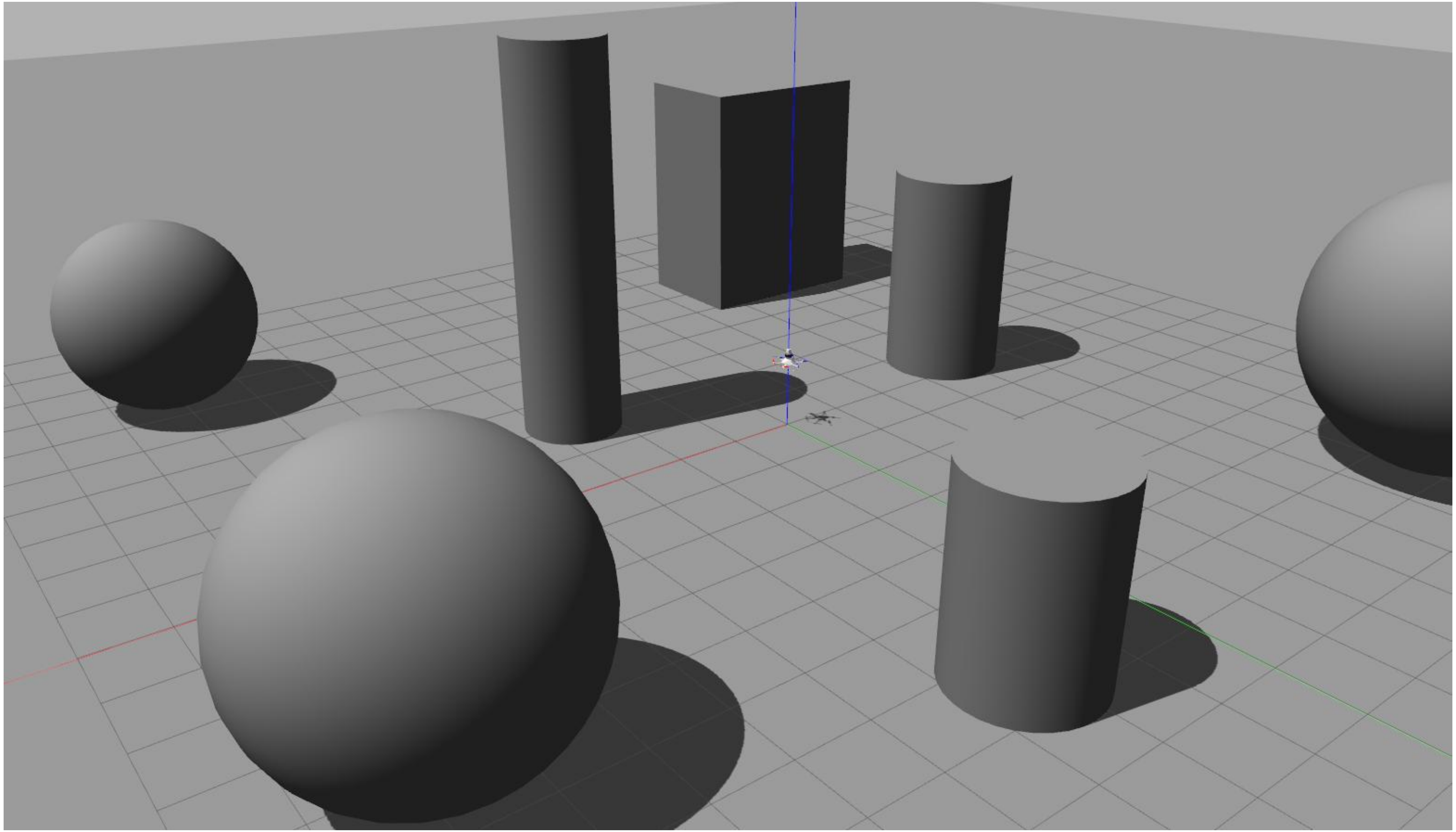
Activity Diagram


Planner
Map Manager
Flight controller
CAMERA
YOLO NN
OUTPUT TO USER
Start
Update State
Subsequent Iterations
First Iteration
Spawn New Nodes
Remove Colliding Nodes
Check Collision for Edges
Remove Colliding Edges
Find Djikstra Shortest path in graph
Plan Next Waypoint
Destination Reached?
No
Yes
Collect data from LIDAR
Update Map
Check Collision for Nodes
Check Collision Status of Voxels using updated map
Determine Voxels representing given entitiy
Check Collision for Edges
Detect obstacles in path
Obstacle detected?
No
Yes
command for next waypoint
Send alert
weighted history of drone states
EKF
Current Waypoint
Current State Estimate
Compute desired pose
Compute Actuator Commands
Publish motor commands
Get new state from Gazebo
capture image
convert to ROS message
publish on /image_raw
Pass through NN
check
dump the image message
create a bounding box around object
Subscribe to /img_raw
Keep messages in queue
Publish detected image on respective topic
Save at latest updated location

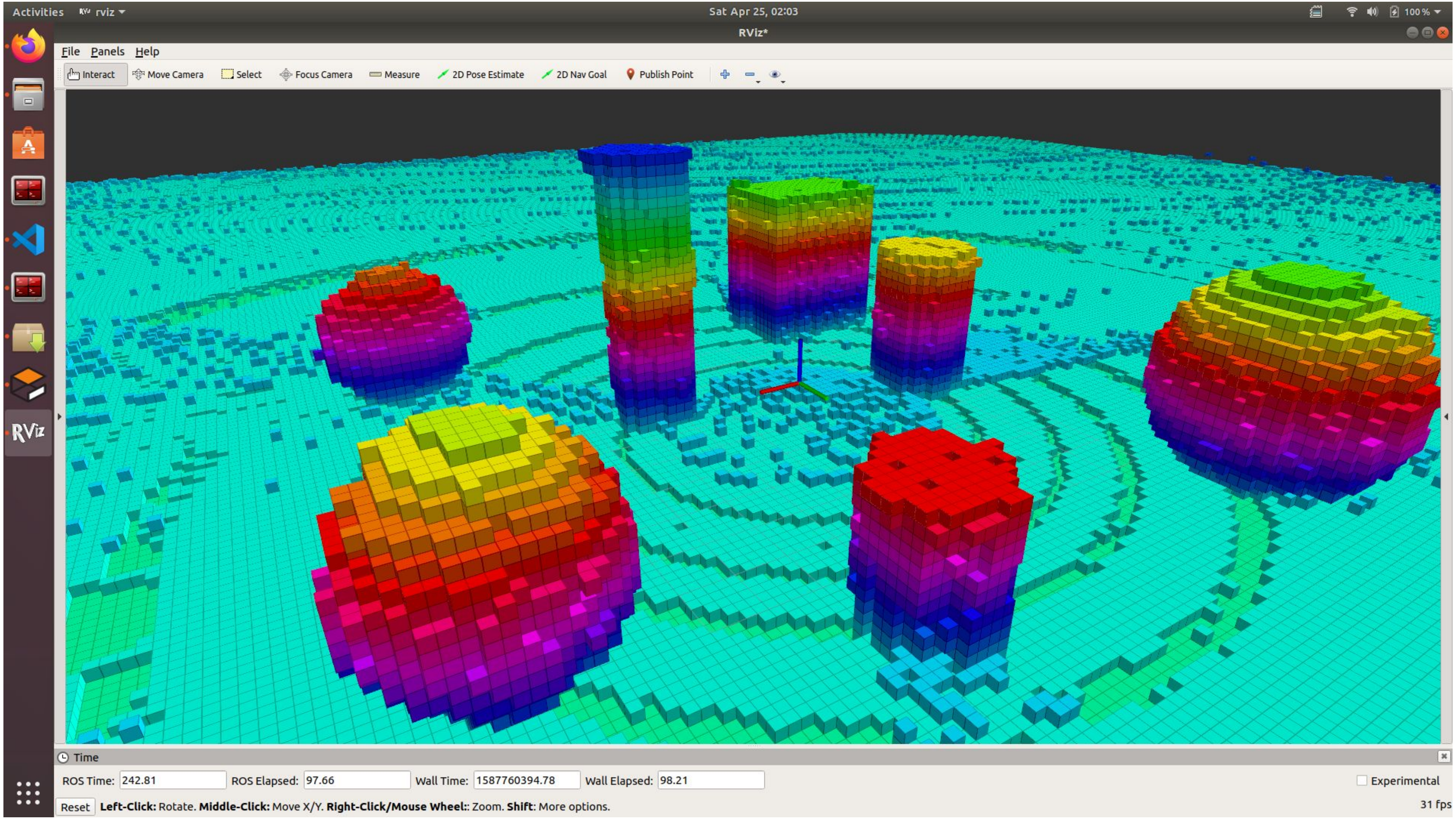
Timing Diagram



Data Visualisation



Data Visualisation



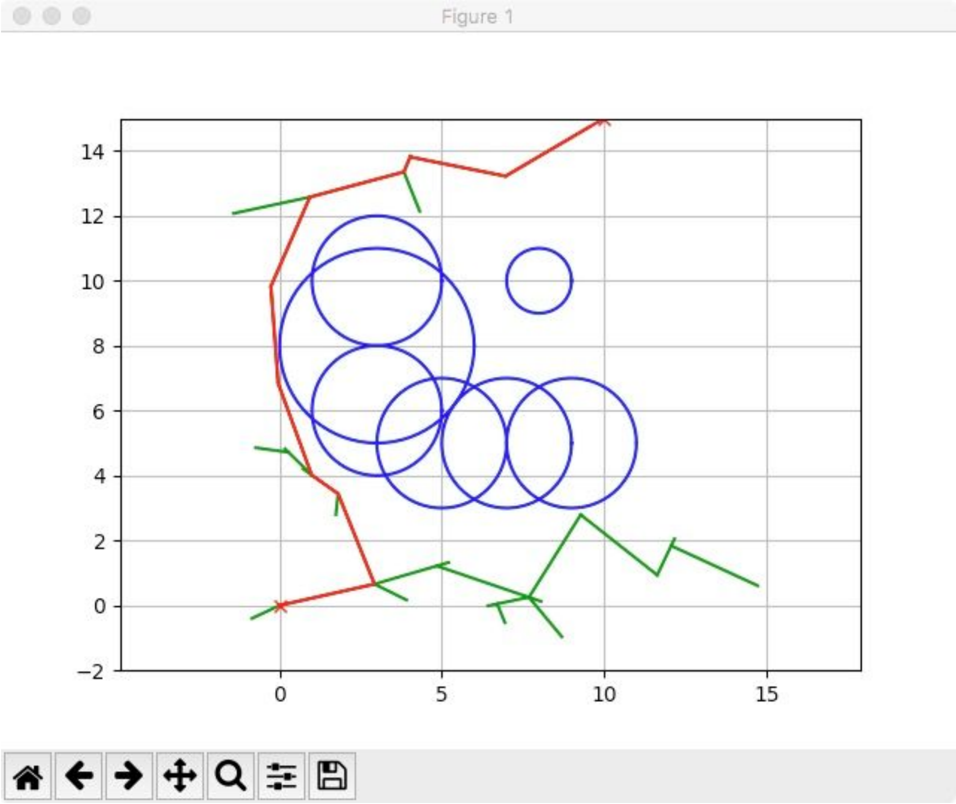
ROS Time: 242.81 ROS Elapsed: 97.66 Wall Time: 1587760394.78 Wall Elapsed: 98.21

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click/Mouse Wheel:**: Zoom. **Shift**: More options.

RRT (Rapidly exploring Random Tree)

- An algorithm used to plot a path from a starting to an ending point, while avoiding obstacles
- Path is not necessarily optimal
- A random node is generated in code, and it is checked if that point lies inside an object.
- It is then chained to the nearest node, ensuring that chain does not hit obstacles. The chain is appended to the graph.
- Path generation ends once a node is generated within a target radius around final point
- I have written a python implementation for RRT, which takes input for ending point. I have plotted some obstacles, which can be modified :

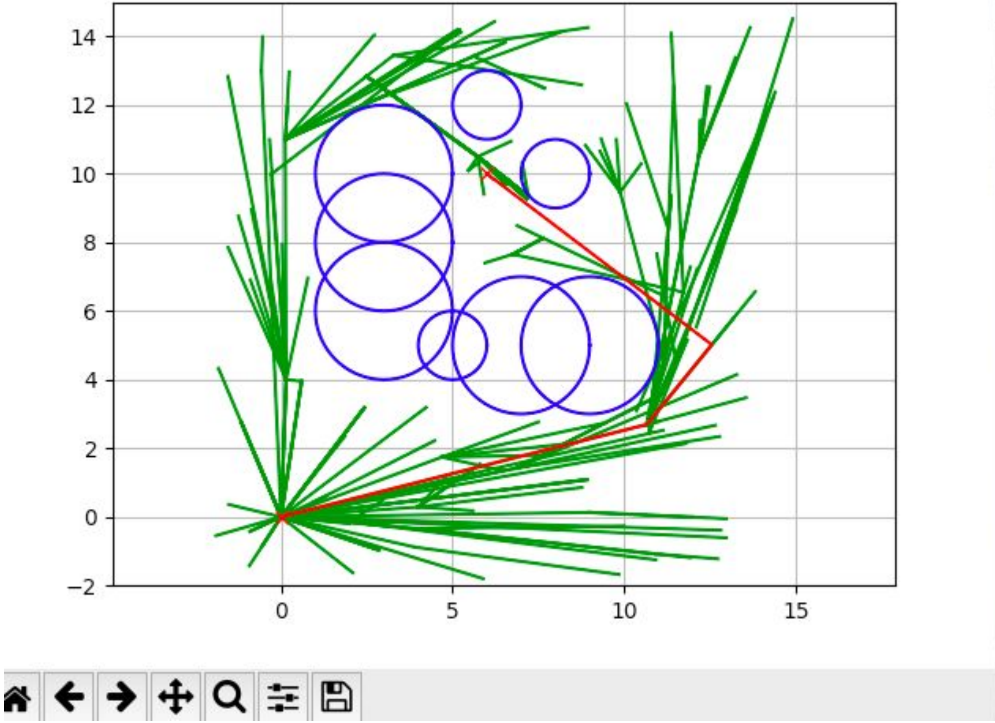
Taking x y=10 15

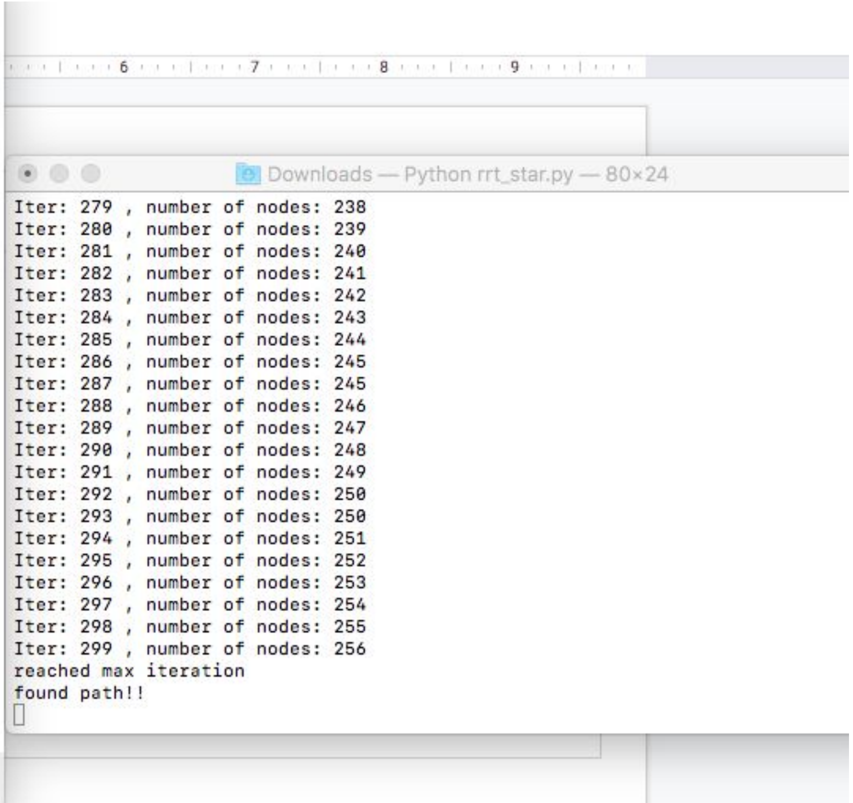


RRT*

- modified version of RRT, used to produce an optimal path
- After a large number of nodes has been plotted, the shorted path is generated
- 1st modification: RRT* records the distance each vertex has traveled relative to its parent vertex= cost()of the vertex. After the closest node is found in the graph, a neighborhood of vertices in a fixed radius from the new node are examined. If a node with a cheaper cost() than the proximal node is found, the cheaper node replaces the proximal node.
- The second difference RRT* adds is the rewiring of the tree. After a vertex has been connected to the cheapest neighbor, the neighbors are again examined. Neighbors are checked if being rewired to the newly added vertex will make their cost decrease. If the cost does indeed decrease, the neighbor is rewired to the newly added vertex. This feature makes the path more smooth.

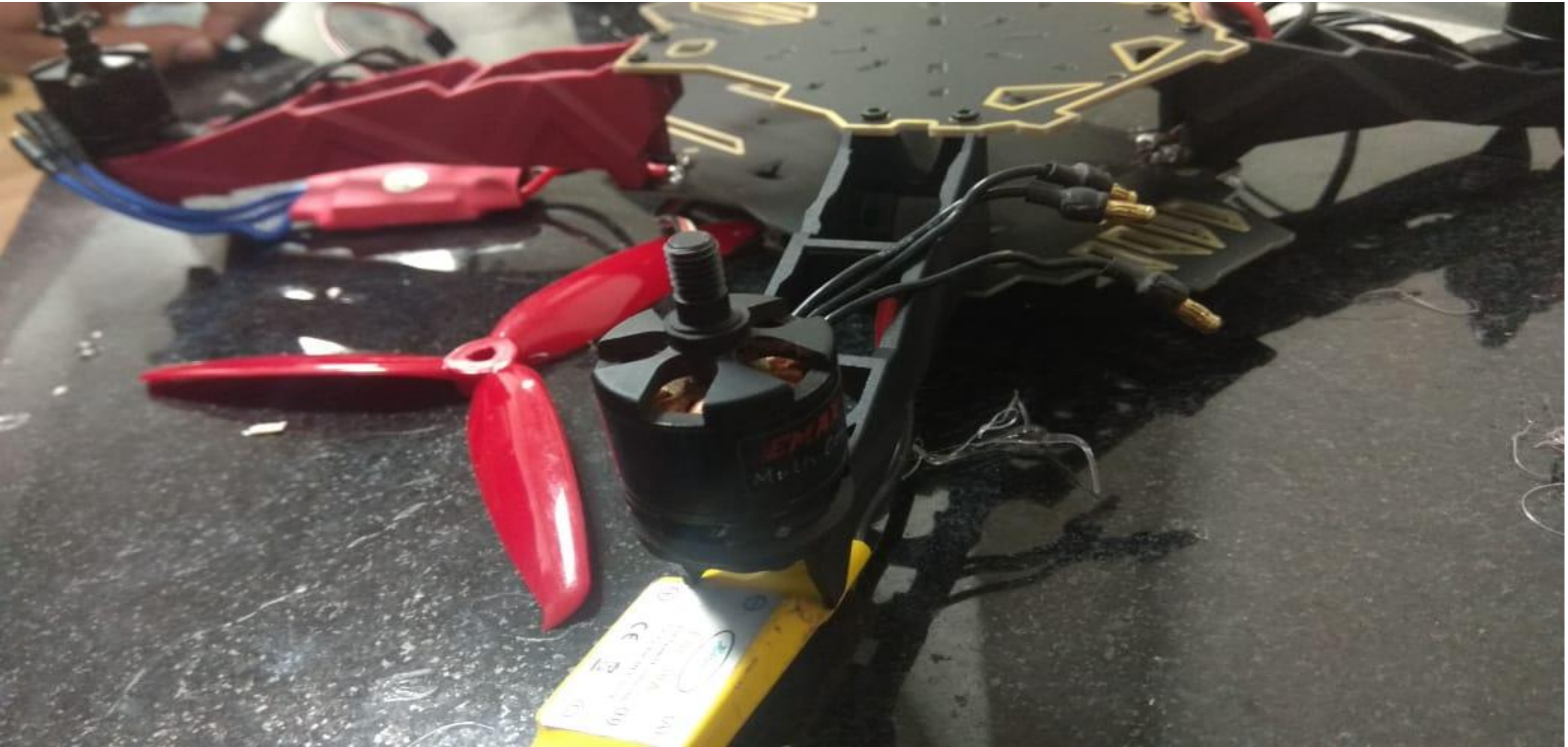
RRT* implementation with 300 iterations:





```
Iter: 279 , number of nodes: 238
Iter: 280 , number of nodes: 239
Iter: 281 , number of nodes: 240
Iter: 282 , number of nodes: 241
Iter: 283 , number of nodes: 242
Iter: 284 , number of nodes: 243
Iter: 285 , number of nodes: 244
Iter: 286 , number of nodes: 245
Iter: 287 , number of nodes: 245
Iter: 288 , number of nodes: 246
Iter: 289 , number of nodes: 247
Iter: 290 , number of nodes: 248
Iter: 291 , number of nodes: 249
Iter: 292 , number of nodes: 250
Iter: 293 , number of nodes: 250
Iter: 294 , number of nodes: 251
Iter: 295 , number of nodes: 252
Iter: 296 , number of nodes: 253
Iter: 297 , number of nodes: 254
Iter: 298 , number of nodes: 255
Iter: 299 , number of nodes: 256
reached max iteration
found path!!
```

Real world assembled drone



Rapidly Exploring Random Graph (RRG) Based Planner

Features

- Undirected graph generated with cycles
- Multiple paths between two points
- Each node is connected to all nodes within the specified radius
- Asymptotic probability of converging to the optimal path is 1
- Can be generalised across robots without changes unlike some reinforcement learning models

Rapidly Exploring Random Graph (RRG) Based Planner

Algorithm

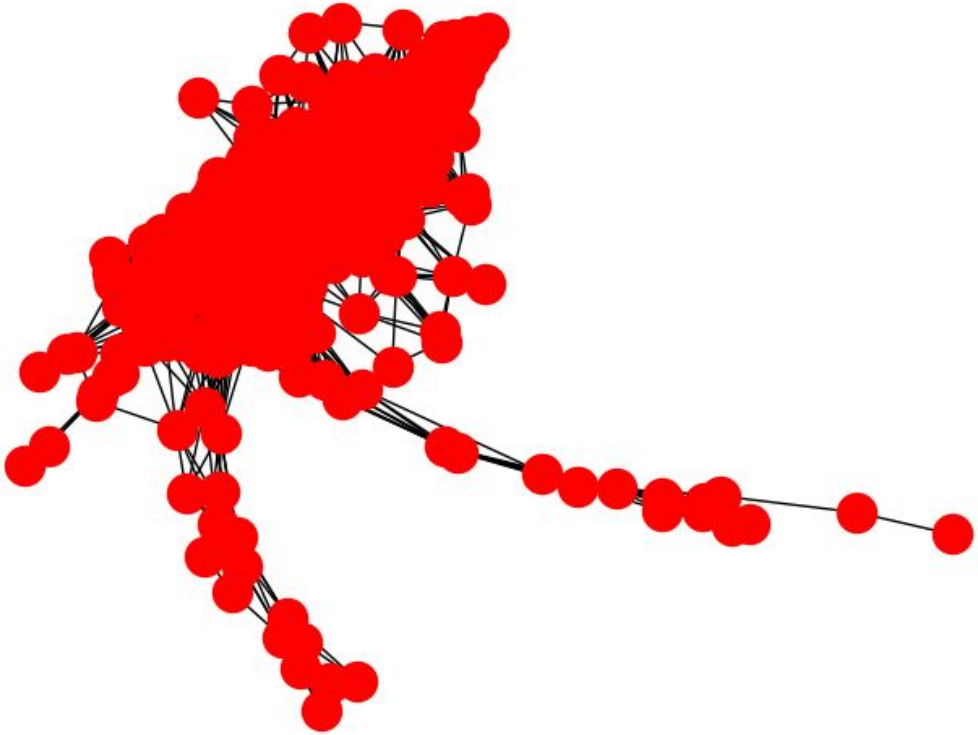
- Spawn random nodes in space
- Check nodes for obstacles at the same point
- If obstacles found, remove the node
- Add edges from newly spawned node to all nodes within a radius
- Check edges for obstacles along path between two nodes
- Remove edges in colliding paths
- Repeat till the specified number of points are generated
- Run Dijkstra’s algorithm to find the shortest path to the target region

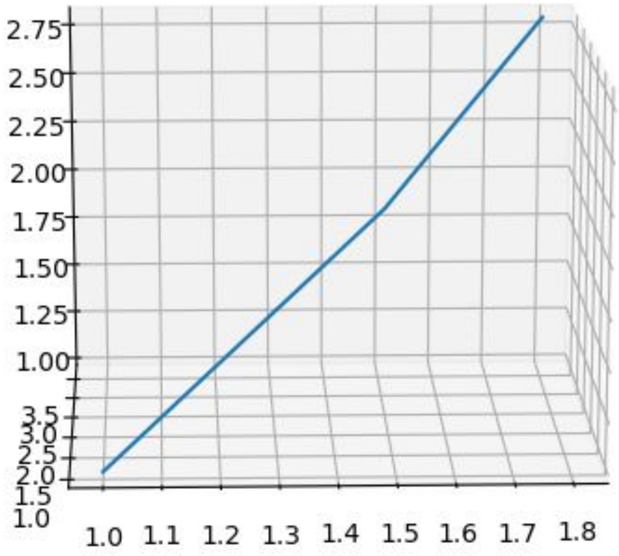
Rapidly Exploring Random Graph (RRG) Based Planner

Python Implementation

- The RRG graph is created using the NetworkX library
- The algorithm executes as a ROS node in the ses_planner package
- A uniform distribution is used to spawn nodes
- A granularity parameter is passed to specify the minimum interval between points
- Two service calls are made to the map manager to check nodes and edges for collision (/check_boxes and /check lines)
- The dijkstra_path function is used to find the shortest path and the waypoints are returned as a list of Point messages.

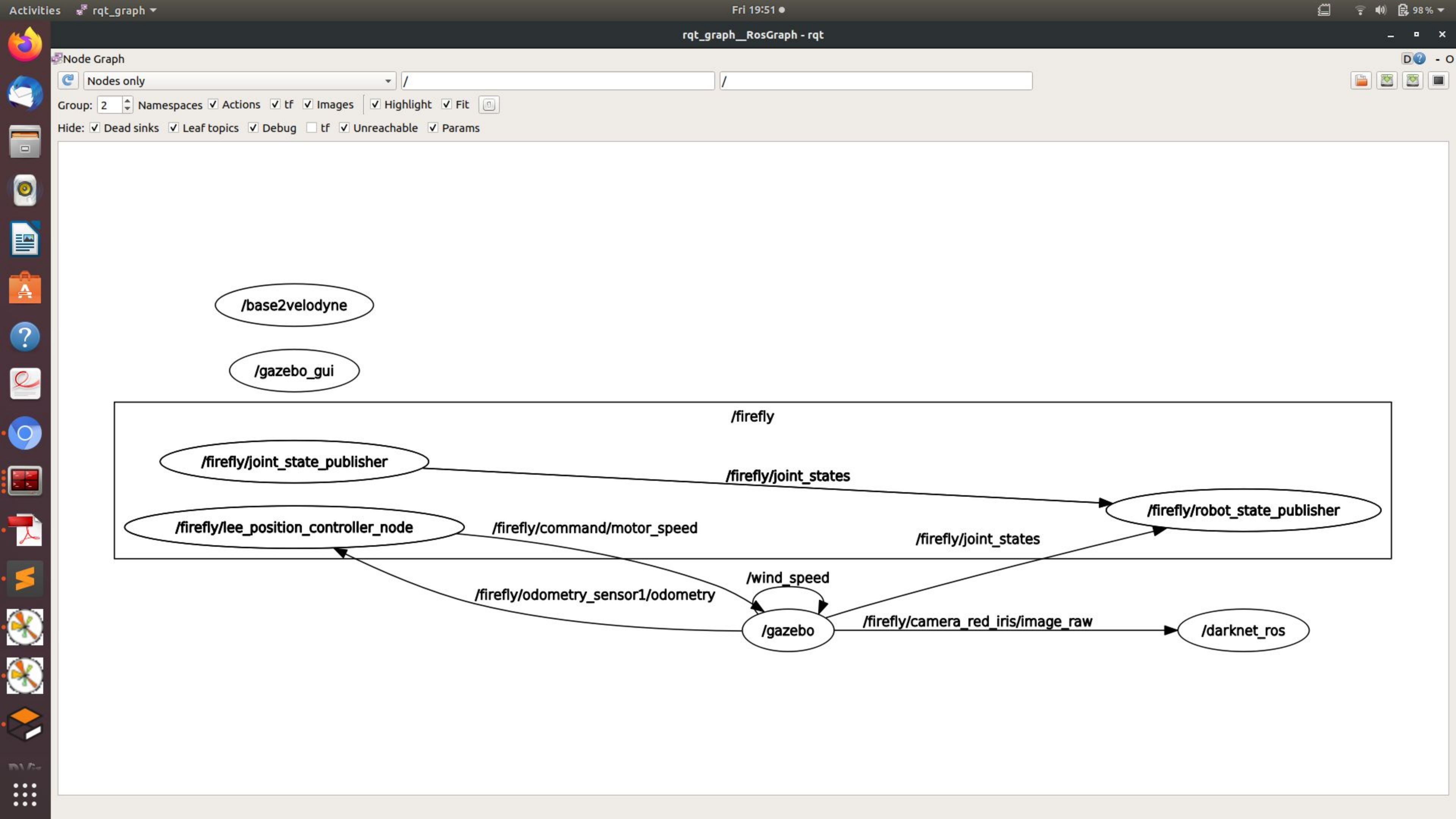
RRG - Theoretical graph and Shortest Path





YOLO-You Only Look Once

Object Detection Technique



Activities rqt_graph Fri 19:51 98 %

rqt_graph__RosGraph - rqt

Node Graph

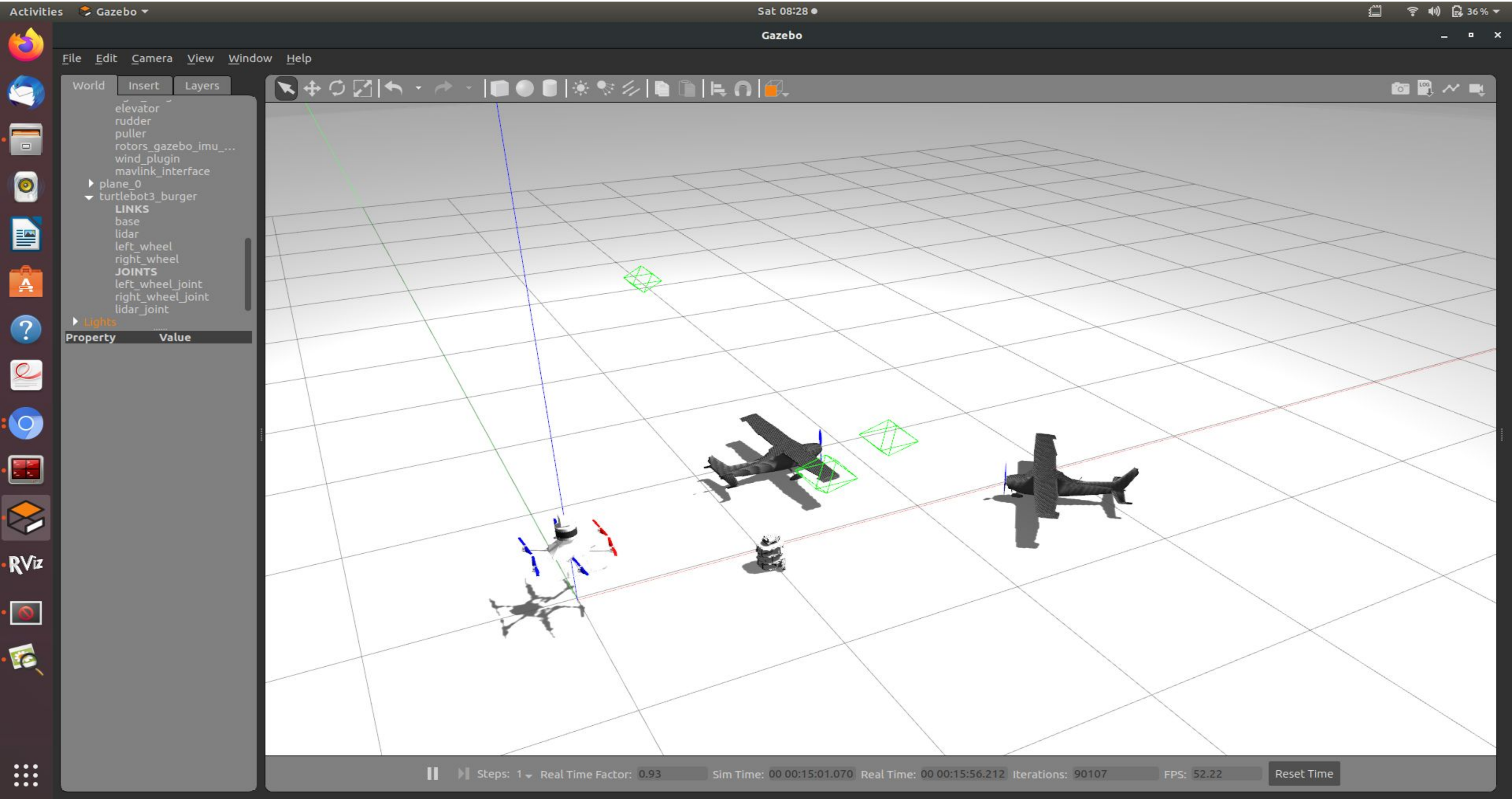
Nodes only / /

Group: 2 Namespaces ☑ Actions ☑ tf ☑ Images ☑ Highlight ☑ Fit

Hide: ☑ Dead sinks ☑ Leaf topics ☑ Debug □ tf ☑ Unreachable ☑ Params

/base2velodyne
/gazebo_gui
/firefly
/firefly/joint_state_publisher
/firefly/joint_states
/firefly/robot_state_publisher
/firefly/lee_position_controller_node
/firefly/command/motor_speed
/firefly/joint_states
/wind_speed
/firefly/odometry_sensor1/odometry
/gazebo
/firefly/camera_red_iris/image_raw
/darknet_ros

Third Person View

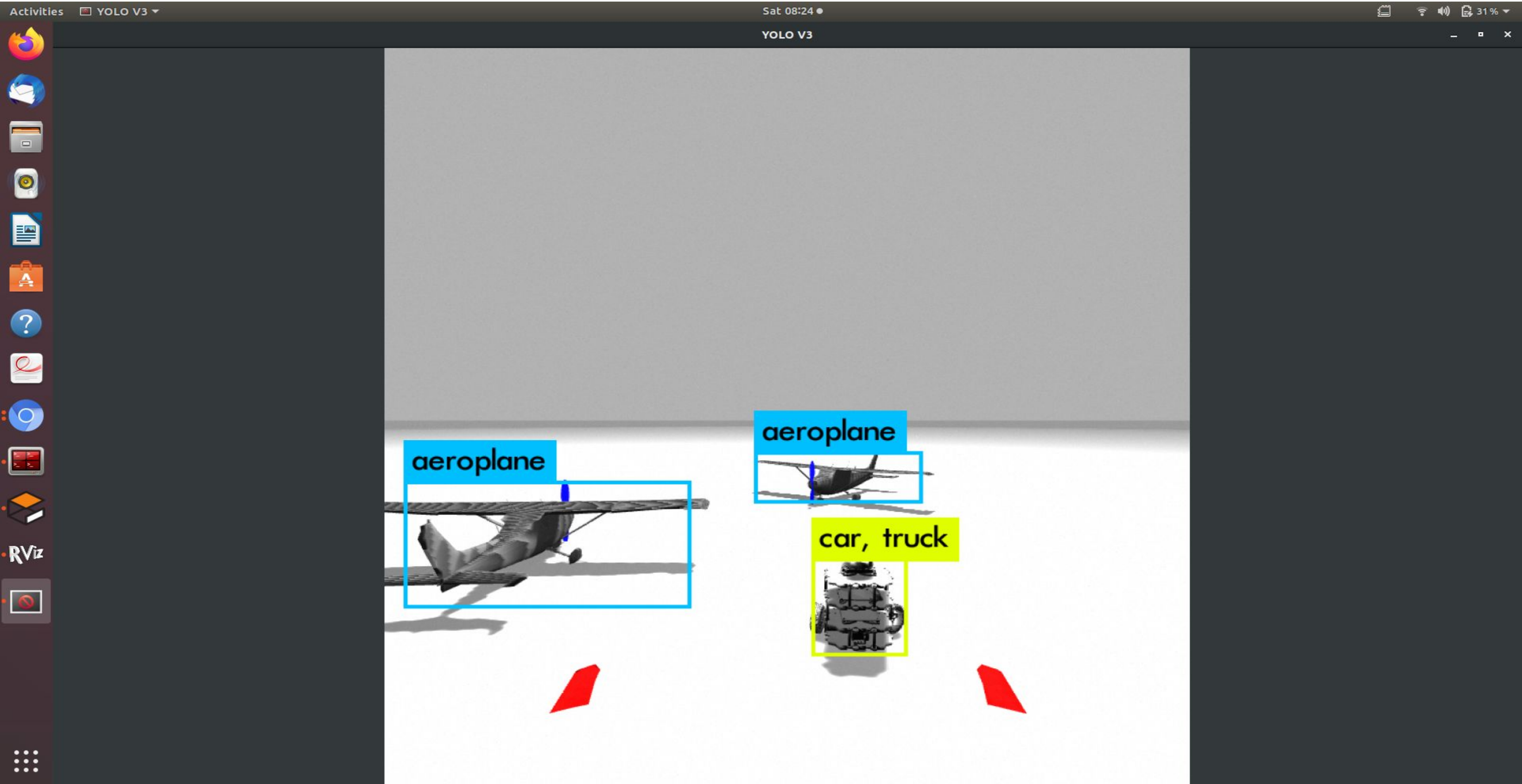


elevator
rudder
puller
rotors_gazebo_imu_...
wind_plugin
mavlink_interface
plane_0
turtlebot3_burger
LINKS
base
lidar
left_wheel
right_wheel
JOINTS
left_wheel_joint
right_wheel_joint
lidar_joint
Lights

Property	Value

Steps: 1 Real Time Factor: 0.93 Sim Time: 00 00:15:01.070 Real Time: 00 00:15:56.212 Iterations: 90107 FPS: 52.22

First Person View



Relevant topics

Camera Feed : /firefly/camera_red_iris/image_raw

YOLO topics:/darknet_ros/bounding_boxes

/darknet_ros/check_for_objects/cancel

/darknet_ros/check_for_objects/feedback

/darknet_ros/check_for_objects/goal

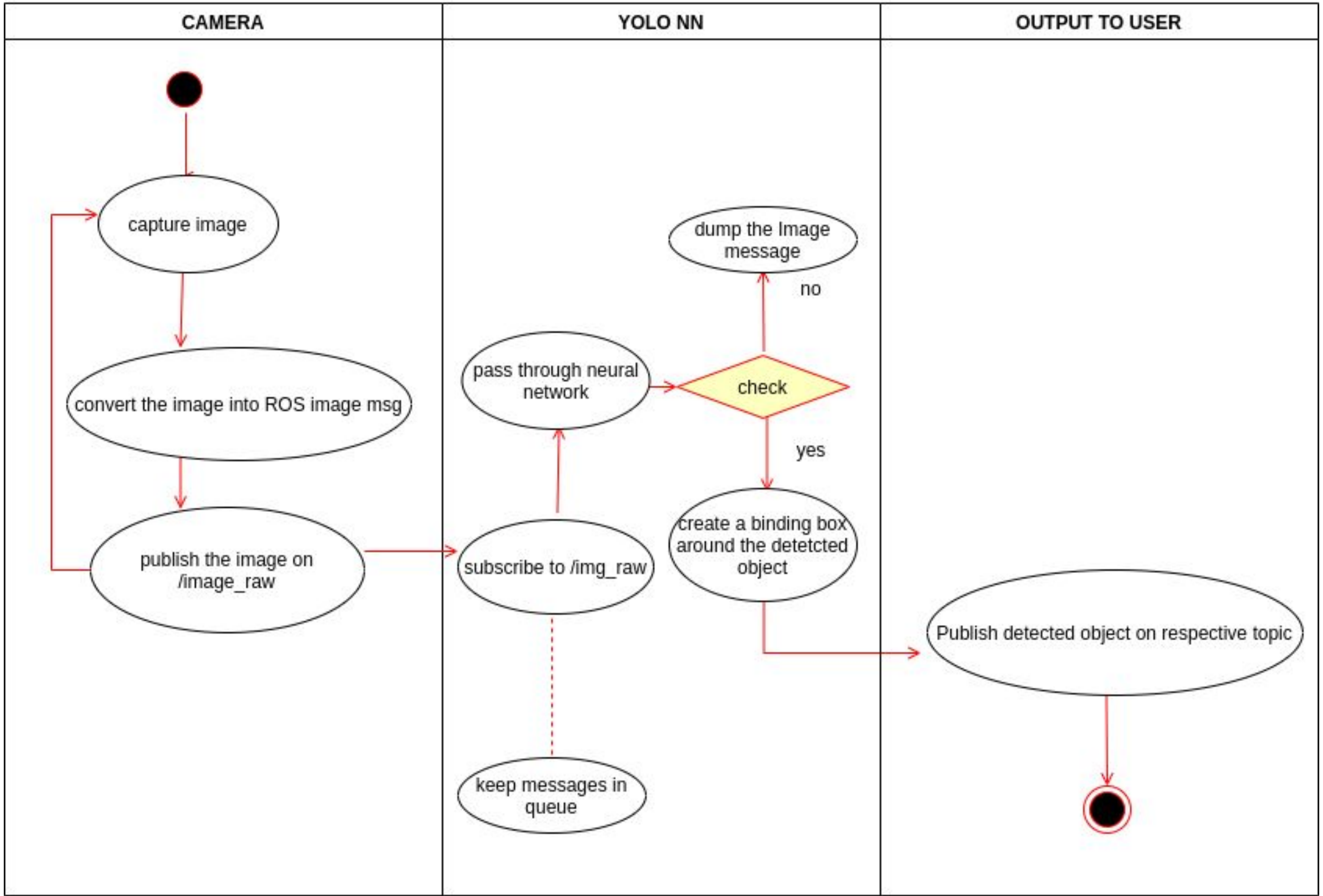
/darknet_ros/check_for_objects/result

/darknet_ros/check_for_objects/status

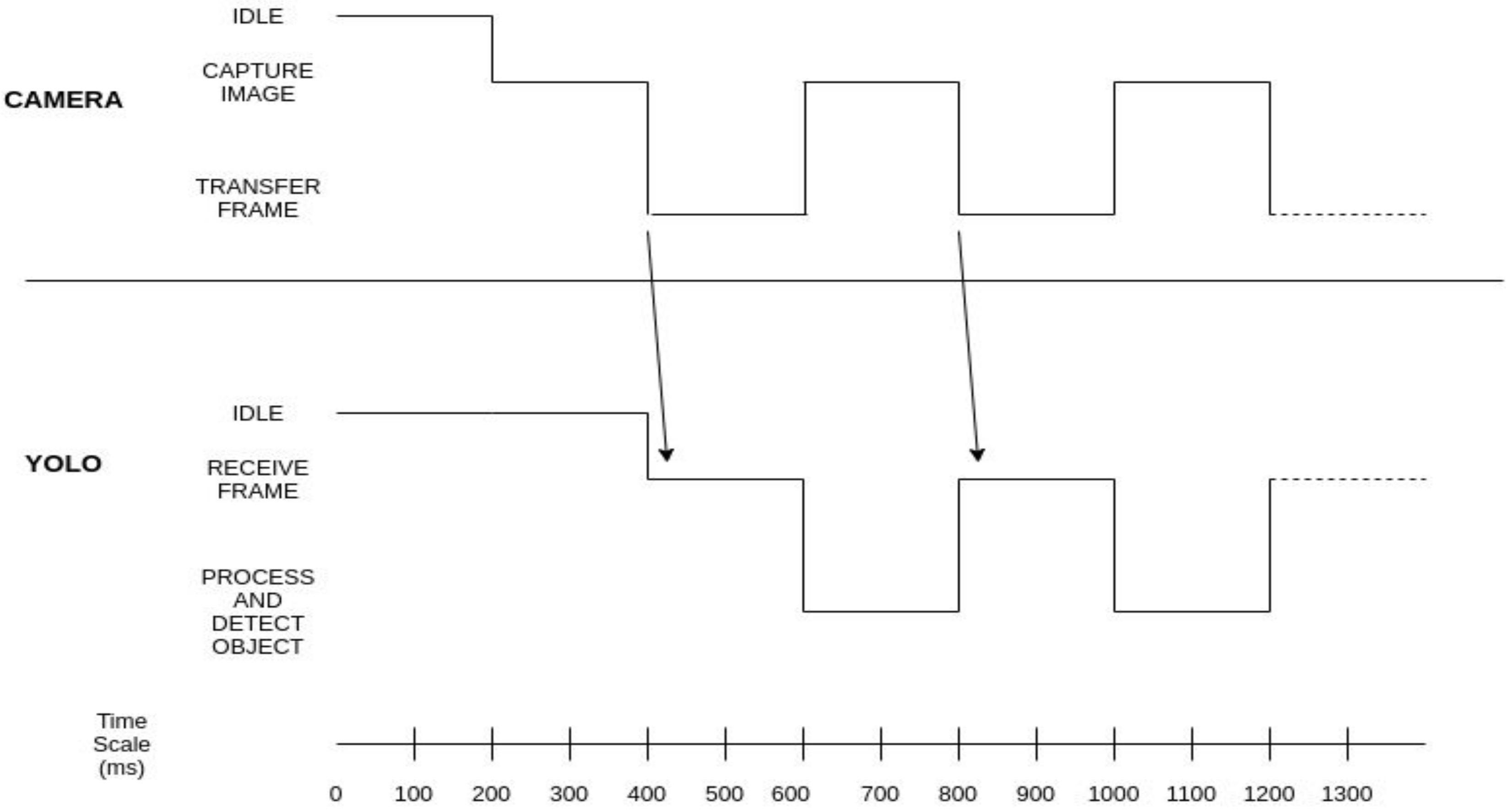
/darknet_ros/detection_image

/darknet_ros/found_object

ACTIVITY DIAGRAM FOR OBJECT DETECTION USING YOLO


CAMERA
YOLO NN
OUTPUT TO USER
capture image
convert the image into ROS image msg
publish the image on /image_raw
subscribe to /img_raw
keep messages in queue
pass through neural network
check
no
dump the Image message
yes
create a binding box around the detetcted object
Publish detected object on respective topic

Timing Diagram for YOLO



ROTORS_SIMULATOR

MAV gazebo simulation platform

What is rotors ?

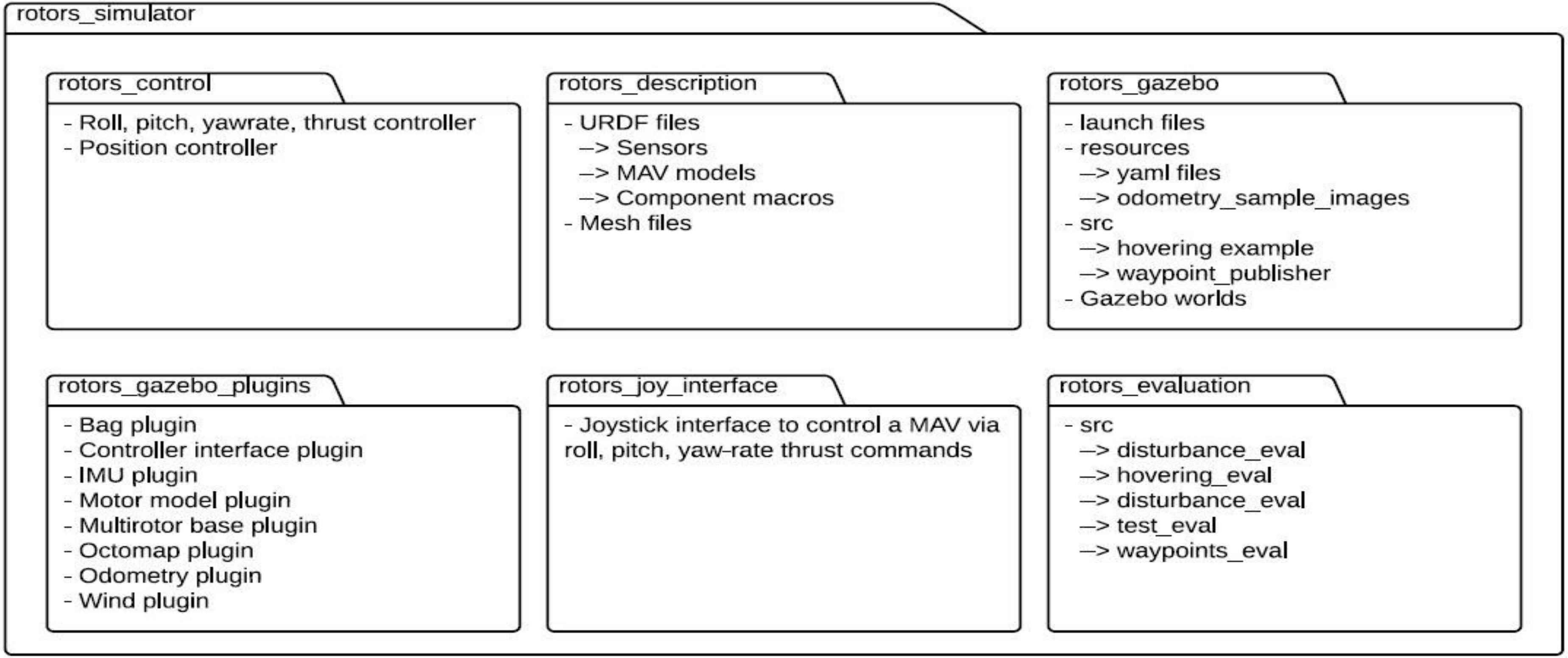
- RotorS is a MAV(Miniature Aerial Vehicles) gazebo simulator. It provides some multirotor models such as the AscTec hummingbird the AscTec Pelican, or Asctec firefly
- All components are designed to be analogous to its real world counterparts.

rotors_simulator

Features-The simulation framework is a good starting point to tackle following higher level tasks on a mav:-

- collision avoidance
- path planning
- Attachment and testing of different sensors and algorithm related to MAVs
- vision based problems, like Simultaneous Localization and Mapping (SLAM)

List of rotors_simulator packages



rotors_simulator

rotors_control

- Roll, pitch, yawrate, thrust controller
- Position controller

rotors_description

- URDF files
  –> Sensors
  –> MAV models
  –> Component macros
- Mesh files

rotors_gazebo

- launch files
- resources
  –> yaml files
  –> odometry_sample_images
- src
  –> hovering example
  –> waypoint_publisher
- Gazebo worlds

rotors_gazebo_plugins

- Bag plugin
- Controller interface plugin
- IMU plugin
- Motor model plugin
- Multirotor base plugin
- Octomap plugin
- Odometry plugin
- Wind plugin

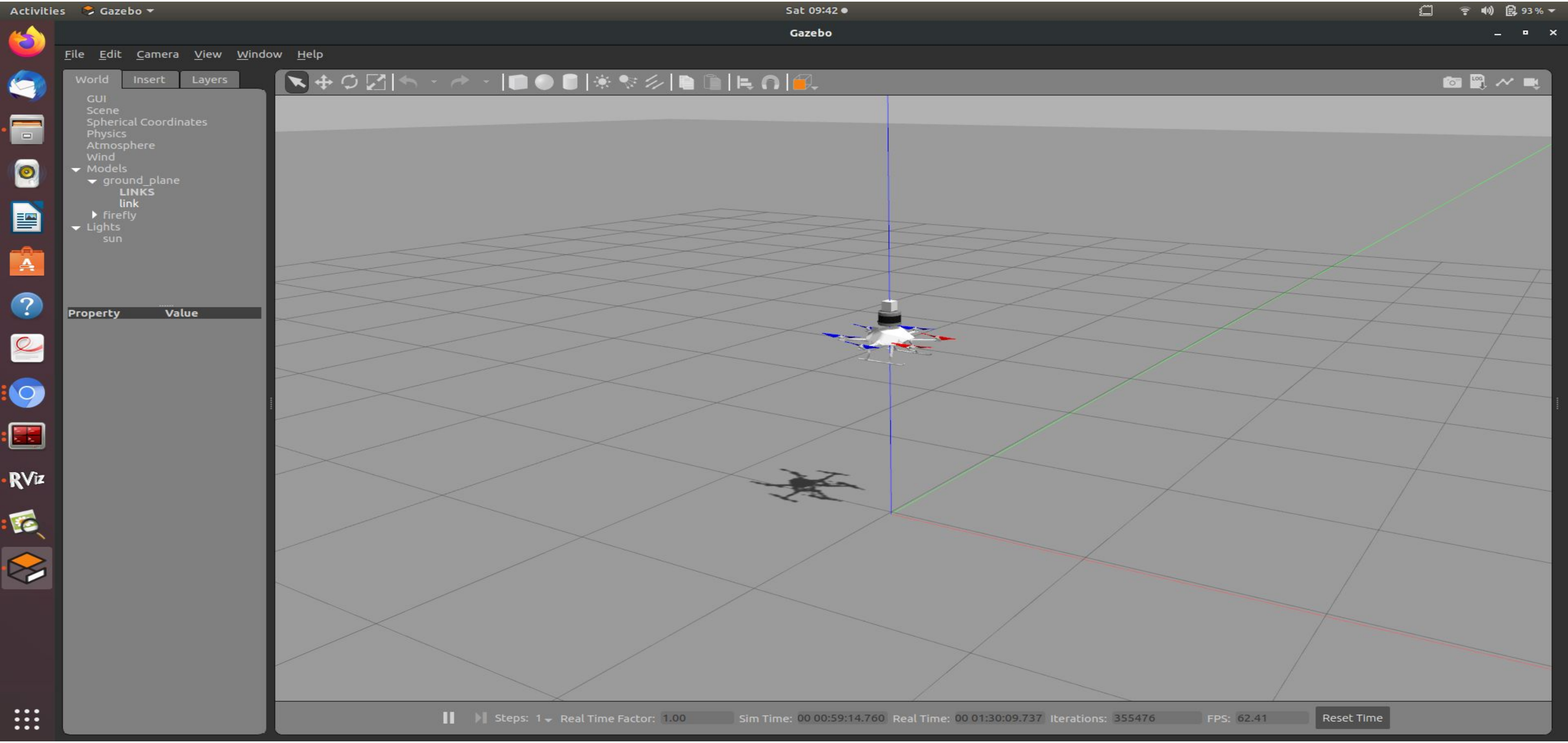
rotors_joy_interface

- Joystick interface to control a MAV via roll, pitch, yaw-rate thrust commands

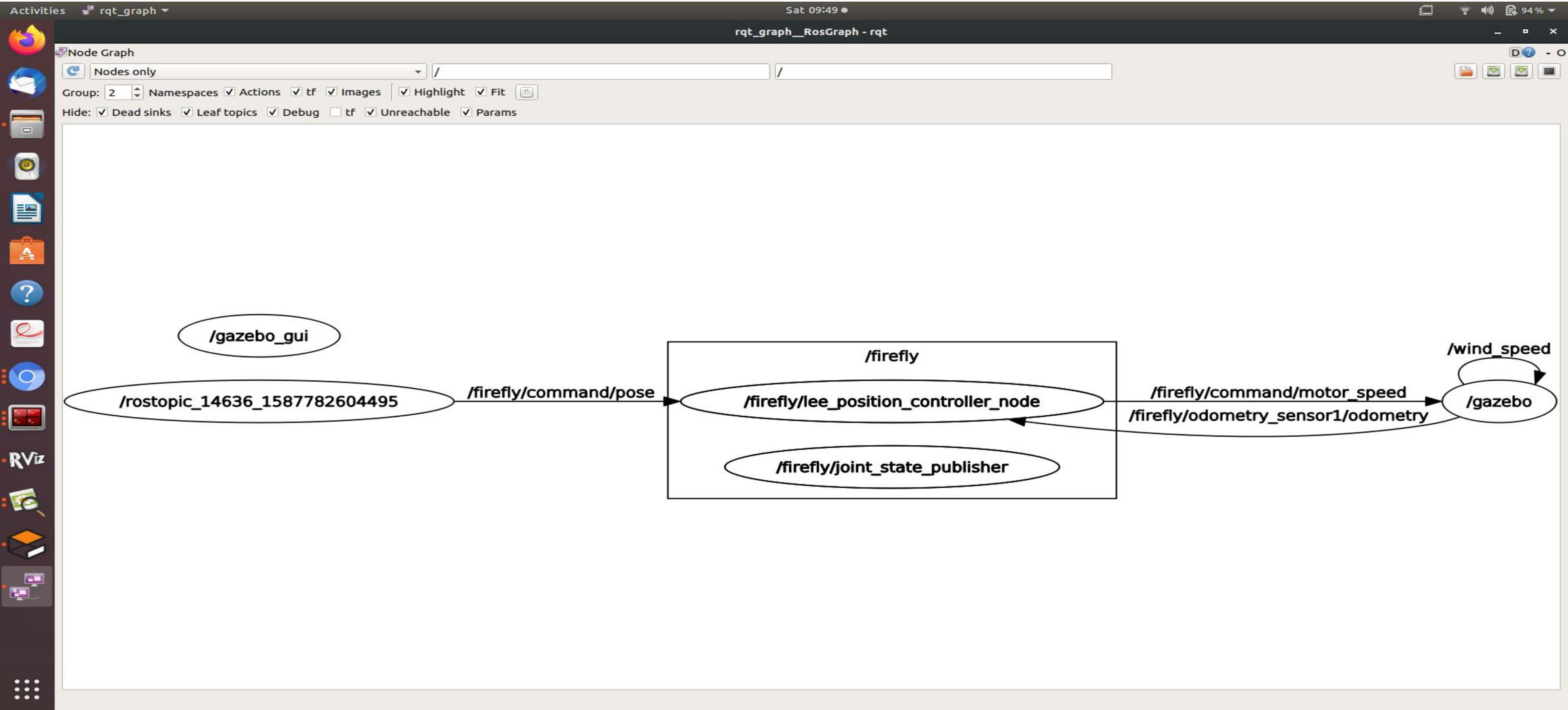
rotors_evaluation

- src
  –> disturbance_eval
  –> hovering_eval
  –> disturbance_eval
  –> test_eval
  –> waypoints_eval

A simulated firefly with a velodyne lidar attached to it



Rqt_graph



rqt_graph__RosGraph - rqt

Node Graph

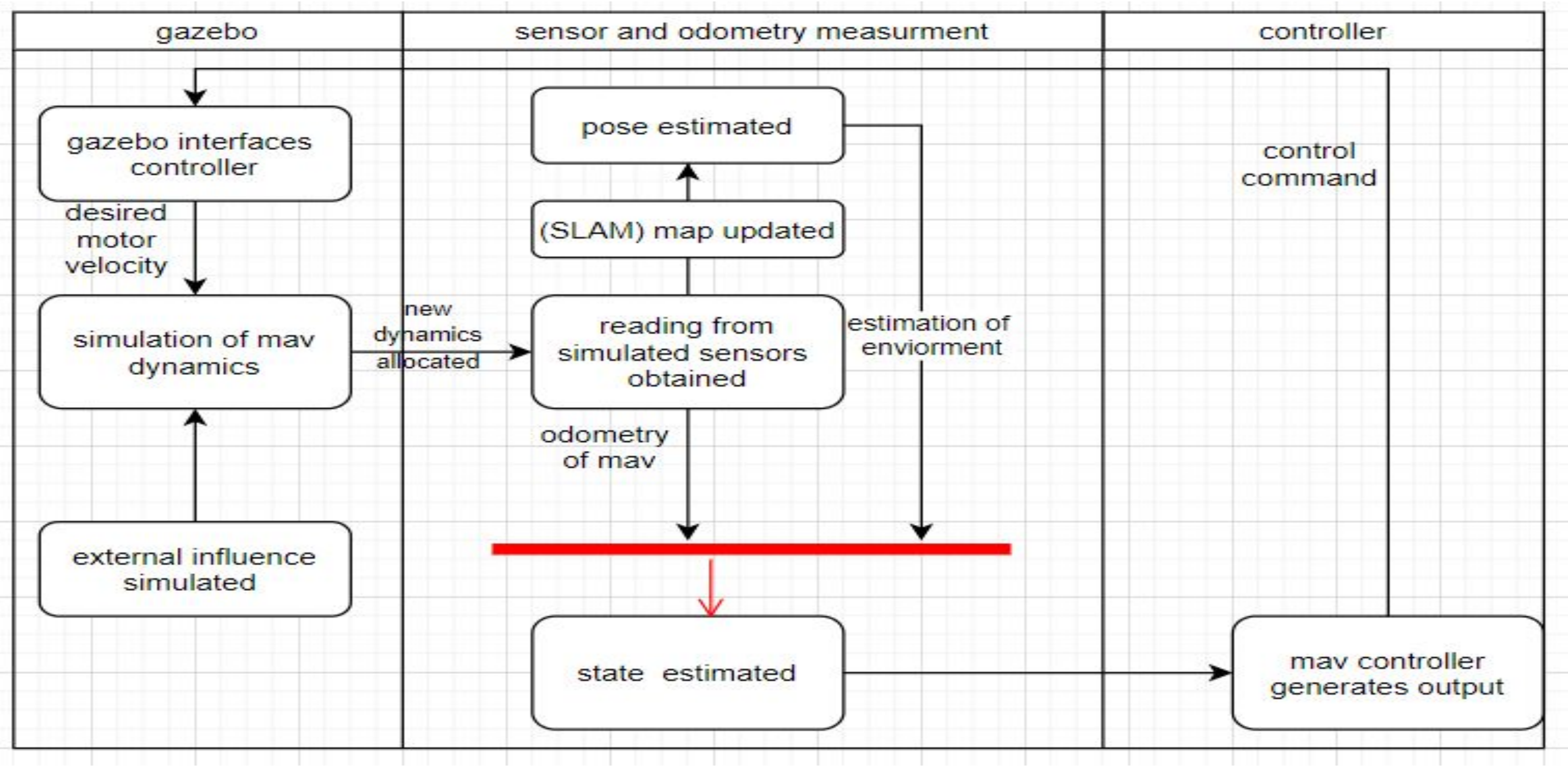
Nodes only ▾ | / | /

Group: 2 ☑ Namespaces ☑ Actions ☑ tf ☑ Images ☑ Highlight ☑ Fit

Hide: ☑ Dead sinks ☑ Leaf topics ☑ Debug □ tf ☑ Unreachable ☑ Params

Activities
rqt_graph
Sat 09:49
94 %
/gazebo_gui
/rostopic_14636_1587782604495
/firefly/command/pose
/firefly
/firefly/lee_position_controller_node
/firefly/joint_state_publisher
/firefly/command/motor_speed
/firefly/odometry_sensor1/odometry
/wind_speed
/gazebo
RViz

Activity diagram


gazebo
sensor and odometry measurment
controller
gazebo interfaces controller
desired motor velocity
simulation of mav dynamics
external influence simulated
new dynamics allocated
pose estimated
(SLAM) map updated
reading from simulated sensors obtained
estimation of enviorment
odometry of mav
state estimated
control command
mav controller generates output