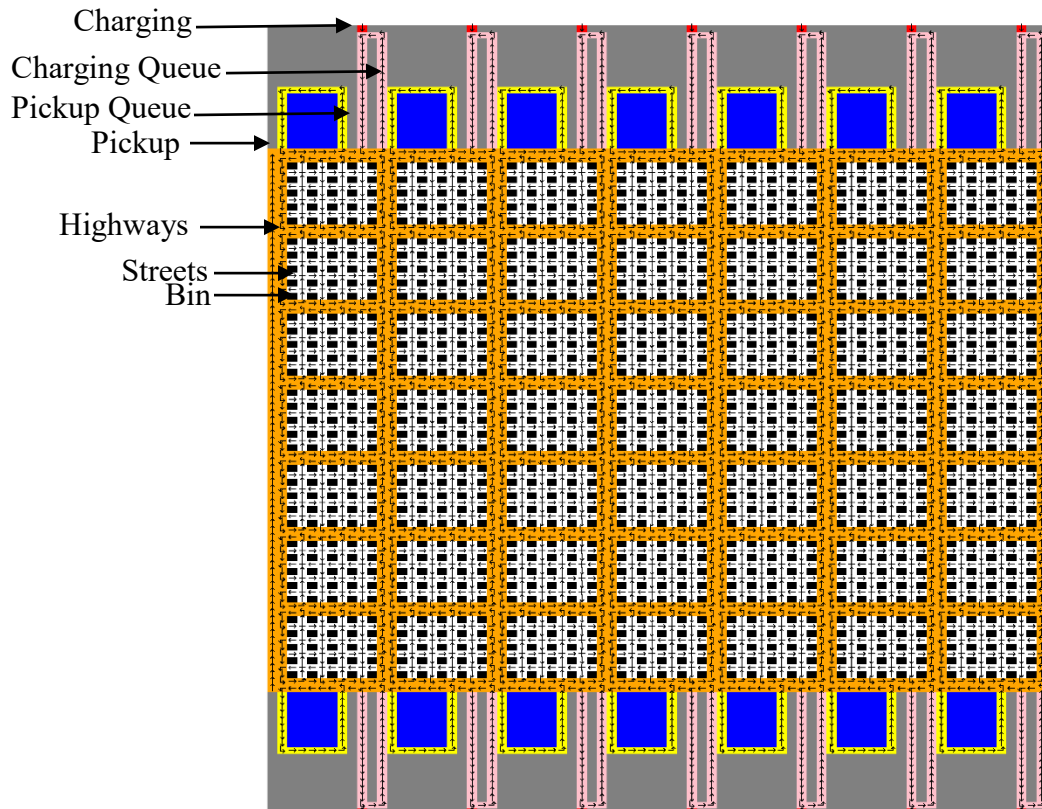


Sorting Robot

Sorting Robot sorts the parcel into their respective bins depending on the locations using barcodes. Map for sorting robots includes parcel bin, streets and highways for robot movements, robot charging points, parcel pickup points where human will place parcel conveyor belt to robot top. Once parcel is placed on robot top, robot should place the parcel to its respective bin by following best possible path and avoiding the congestion as much as possible.

Navigation Graph



```

classDiagram
    class SEARCH_NODE {
        +x - int
        +y - int
        +θ - int
        +COST - double
        +PARENT - search node
    }
    class NAV_GRAPH_NODE {
        +Area Type {street intersection, highway intersection, street road, highway road, bin, obstacle, pickup location, pickup queue area, charger location, next to charger location, charging queue area}
        +Navigation Left Allowed: bool
        +Navigation Right Allowed: bool
        +Navigation Up Allowed: bool
        +Navigation Down Allowed: bool
    }
    class ROBOT {
        +State - as per documentation
        +x - int
        +y - int
        +θ - int
        +pickupID - int
        +dropOffLocation - (x,y)
        +parcelID - int
        +chargerID - int
        +inChargeQueue - bool
        +inPickupQueue - bool
        +barCodeReading - string
        +Controllers and transition conditions - as per documentation
    }
    class SEARCH_ALGORITHM {
        +GETPATH(cell Source, cell Bin) - after pickup
        +GETPATH(node Source, node goal) - for pickup and charging
        +HEURISTICFUNCTION(cell)
        +EDGECOST(node u, node v)
    }
    class MAP {
        +HEATMAP - double[][]
        +ROBOTOCCUPANCY - bool [][]
        +NAVIGATIONGRAPH - NavGraphNode[][]
        +Initilaize() - Load from file
        +UpdateMaps() - Execute Infinitely
        +PositionCallbacks() - from ROS
    }
    class ENVIRONMENT {
        +ADD X
        +DELETE X
        +INITIALIZE X
        +UPDATE X
        X = all entities
    }
    class TRAFFIC_LIGHT {
        +POS x - int
        +POS y - int
        +STATE {as per documentation}
        +TIMER - time in the current state
        +TYPE {HIGHWAY, STREET RD, RU, LD, LU}
        +UPDATE() - called continuously
        +INITIALIZE()
    }
    class CHARGING_MANAGER {
        +ID - int
        +X - int
        +Y - int
        +STATE {reserved, not-reserved}
        +RESERVE()
        +UNRESERVE()
    }
    class DROPOFF_MANAGER {
        +DROPPARCELCOUNT - int
        +DROPOFFAVALIABLE - int
        +ID - int
        +X - int
        +Y - int
        +INIT()
        +INCREASEPARCELCOUNT()
        +RESERVE()
        +UNRESERVE()
    }
    class PICKUP_MANAGER {
        +STATE : {Open, Close}
        +HASMOREPARCELS : bool
        +ROBOTQUEUE - Queue<node>
        +OPENSTATE()
        +CLOSESTATE()
        +ADDEROBOTTOQUEUE()
        +DELETEROBOTINQUEUE()
        +TopRobotInQueue()
        +GetRobotQueueSize()
    }
    class CHARGING_QUEUE_MANAGER {
        +ENQUEUE()
        +DEQUEUE()
        +FRONT()
        +SIZE()
    }
    class PARCEL_MANAGER {
        +IDOFROBOT - int
        +DROPOFF LOCATION - X,Y
        +PICKUPLLOCATION - int
        +WAITINGTIME - double
        +SERVICETIME - double
        +ORDEROfPARCEL - int
        +INIT()
    }
    class ROBOT_QUEUE_NODE {
        +ID - int
    }

    SEARCH_NODE --> SEARCH_ALGORITHM
    NAV_GRAPH_NODE --> MAP
    MAP --> ENVIRONMENT
    TRAFFIC_LIGHT --> ENVIRONMENT
    CHARGING_MANAGER --> ENVIRONMENT
    DROPOFF_MANAGER --> ENVIRONMENT
    PICKUP_MANAGER --> ENVIRONMENT
    CHARGING_QUEUE_MANAGER --> ENVIRONMENT
    PARCEL_MANAGER --> ENVIRONMENT
    ENVIRONMENT --> ROBOT_QUEUE_NODE
    ROBOT --> ENVIRONMENT
    ROBOT_QUEUE_NODE --> PICKUP_MANAGER
    PICKUP_MANAGER --> CHARGING_MANAGER
    CHARGING_MANAGER --> CHARGING_QUEUE_MANAGER
    CHARGING_QUEUE_MANAGER --> CHARGING_MANAGER
    CHARGING_MANAGER --> DROPOFF_MANAGER
    DROPOFF_MANAGER --> CHARGING_MANAGER
    DROPOFF_MANAGER --> PARCEL_MANAGER
    PARCEL_MANAGER --> DROPOFF_MANAGER
    PARCEL_MANAGER --> ROBOT_QUEUE_NODE
    ROBOT_QUEUE_NODE --> PARCEL_MANAGER
    
```

The diagram illustrates the architecture of a ROS-based robot navigation system. It features several key components and their interactions:

- SEARCH NODE**: Contains attributes for position (x, y, θ), cost, and a parent search node.
- NAV GRAPH NODE**: Represents the environment's navigation graph, including area types and navigation permissions.
- ROBOT**: The physical robot, with state, position, orientation, and queue information.
- SEARCH ALGORITHM**: Manages pathfinding, including GETPATH and HEURISTICFUNCTION methods.
- MAP**: Stores environmental data like HEATMAP, ROBOTOCCUPANCY, and NAVIGATIONGRAPH.
- ENVIRONMENT**: The central hub for all entities, managing ADD, DELETE, INITIALIZE, and UPDATE operations.
- TRAFFIC LIGHT**: Manages traffic light states and timers.
- CHARGING MANAGER**: Handles charging station reservations and unreservations.
- DROPOFF MANAGER**: Manages parcel drop-off locations and counts.
- PICKUP MANAGER**: Manages parcel pickup locations and robot queues.
- CHARGING QUEUE MANAGER**: Manages the queue of robots waiting for charging.
- PARCEL MANAGER**: Manages parcel information, including robot ID, location, and timing.
- ROBOT QUEUE NODE**: A node representing a robot in a queue.

Associations and Multiplicities:

- SEARCH NODE** to **SEARCH ALGORITHM**: 1 to 1.
- NAV GRAPH NODE** to **MAP**: 1 to 1.
- MAP** to **ENVIRONMENT**: 1 to 1.
- TRAFFIC LIGHT** to **ENVIRONMENT**: 1 to 1.
- CHARGING MANAGER** to **ENVIRONMENT**: 1 to 1.
- DROPOFF MANAGER** to **ENVIRONMENT**: 1 to 1.
- PICKUP MANAGER** to **ENVIRONMENT**: 1 to 1.
- CHARGING QUEUE MANAGER** to **ENVIRONMENT**: 1 to 1.
- PARCEL MANAGER** to **ENVIRONMENT**: 1 to 1.
- ENVIRONMENT** to **ROBOT_QUEUE_NODE**: 1 to 1.
- ROBOT** to **ENVIRONMENT**: 1 to 1.
- ROBOT_QUEUE_NODE** to **PICKUP_MANAGER**: 1 to 1.
- PICKUP_MANAGER** to **CHARGING_MANAGER**: 1 to 1.
- CHARGING_MANAGER** to **CHARGING_QUEUE_MANAGER**: 1 to 1.
- CHARGING_QUEUE_MANAGER** to **CHARGING_MANAGER**: 1 to 1.
- CHARGING_MANAGER** to **DROPOFF_MANAGER**: 1 to 1.
- DROPOFF_MANAGER** to **CHARGING_MANAGER**: 1 to 1.
- DROPOFF_MANAGER** to **PARCEL_MANAGER**: 1 to 1.
- PARCEL_MANAGER** to **DROPOFF_MANAGER**: 1 to 1.
- PARCEL_MANAGER** to **ROBOT_QUEUE_NODE**: 1 to 1.
- ROBOT_QUEUE_NODE** to **PARCEL_MANAGER**: 1 to 1.

Class Diagram

Heat Map: Heatmap is used for congestion avoidance. Search Algorithm avoids area of high heat value. A star algorithm tries to schedule robot to go through area of lower cost thus distributing map to avoid congestion at particular place.

$$\text{HeatMap}(t) = \eta \Delta t \text{HeatMap}(t - \Delta t) + \text{OccupancyMap}(t)$$

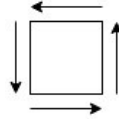
η = evaporation rate (typically 0.75 per millisecond)

Search:

A* Algorithm. It operates on (x, y, θ) as state vector.

There are two variants of the algorithm:

1. Shortest cost path to a bin in which case destination can be anyone of four adjoining edges.



2. It is the case of charging /pickup queue, where goal is available (x_G, y_G, θ_G) representing location of starting of the queue of chosen charging station / pickup point.

- a. Edge Cost:

$$w(u < u_x, u_y, u_\theta >, v < v_x, v, v_\theta >) = \left(\frac{|u_x - v_x| + |u_y - v_y|}{v} + \frac{\text{angleDifference}(u_\theta, v_\theta)}{\omega} \text{penalty} \right) (1 + \alpha \text{heatMap}(v_x, v_y))$$

v =maximum linear speed, ω =maximum angular speed, penalty is applied to penalize stopping and turning (≈ 1.2), α penalizes heat map (to experiment)

- b. Heuristic Cost for bin search:

$$h(u < u_x, u_y, u_\theta >) = \max \left(\frac{|u_x - G_x| + |u_y - G_y|}{v} - \frac{1}{v}, 0 \right)$$

Here (G_x, G_y) stands for the bin location

- c. Heuristic Cost for goal search:

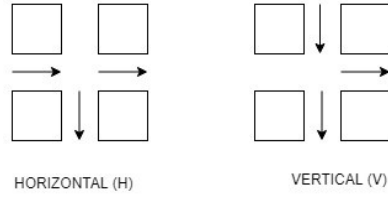
$$h(u < u_x, u_y, u_\theta >) = \frac{|u_x - G_x| + |u_y - G_y|}{v} + \frac{\text{angleDifference}(u_\theta, G_\theta)}{\omega} \text{penalty}$$

Traffic Light

Purpose of Traffic Light Management System is to ensure the robots don't collide **sideways** which is important given that there is no proximity sensor that tells robots on the sides.

The intersection management policy is that at any point of time only one robot will be allowed to center the intersection thus avoiding collision. The traffic light ensures adherence of the policy.

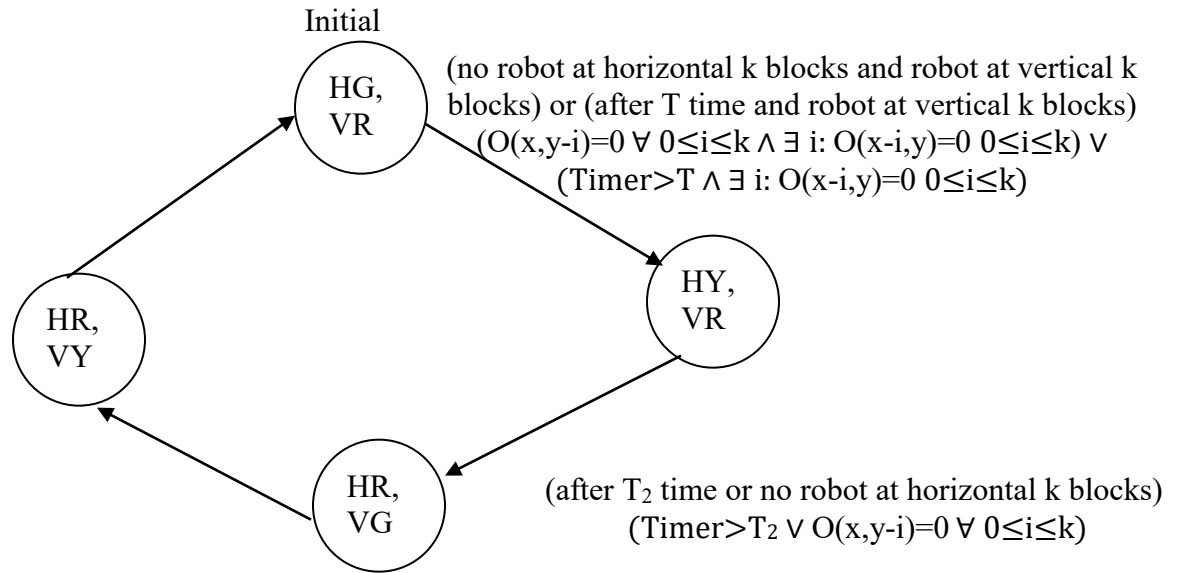
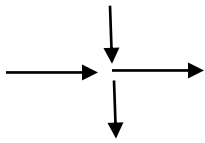
- 1) Traffic light related to the street intersection has 2 states:
 - a) Traffic originating from Horizontal Lane can flow
 - b) Traffic originating from Vertical Lane can flow



Traffic light change is modeled as FSM where vertices represent state of traffic light and the edges represent condition meeting which the state will change.

The FSM representing Traffic Light change is given below:

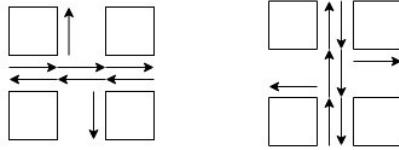
Case RD (right down):



HG, VR: Horizontal Green/Vertical Red
 HY, VR: Horizontal Yellow, Vertical Red
 HR, VG: Horizontal Red, Vertical Green
 HR, VY: Horizontal Red, Vertical Yellow
 RD : Right Down
 O : Occupancy Map

Cases Right Up (RU), Left Down (LD), Left Up (LU) can be similarly enumerated.

2) Traffic Light for Highways: It operates in two states:



Here also Traffic Light condition is exactly same as for Street Traffic Lights.

Coordinate Axis Systems

The navigation map, heat map, search and traffic light use the cell coordinate axis system where origin is top-left, X axis is the vertical axis, Y axis is the horizontal axis and the coordinate system is discretized into cells.

The robot positions and simulator use the real coordinate axis system.

A central utility exists to convert the world coordinate axis system into cell coordinate axis system.

Robot BFSM:

Overall Architecture to control Robot a behavioral FSM. There vertices represent different types of controllers. Robot at any state means the corresponding controller is loaded on the robot for motion. The edges represent the transition condition which are continuously monitored. Upon reaching the transition condition the controller is changed.

Behaviors:

Select Pickup Behaviour

$\text{pickup} = \arg \min w_1 \text{CostToReachPickup}(i) + w_2 \text{PickupQueueSize}(i) : \text{pickup}(i) \text{ has more parcels}$

Select Charger

$\text{pickup} = \arg \min w_1 \text{CostToReachCgarger}(i) + w_2 \text{ChargerQueueSize}(i)$

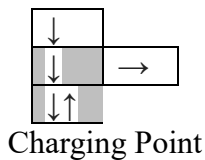
Robot Controller Callbacks to Ensure:

i) Callback when robot enters the pickup area, if operational ($\text{robotInPickupQueue} = \text{False}$ and $\text{RobotInPickupArea} = \text{true}$ and $\text{Pickup}(i)$ has more parcels). If the condition is true, call “enqueue pickup queue” and set $\text{robotInPickupQueue} = \text{True}$

ii) Callback when robot enters the charging area ($\text{robotInChargeQueue} = \text{False}$ and $\text{RobotInChargeArea} = \text{true}$). If the condition is true, call “enqueue charge queue” and set $\text{robotInChargeQueue} = \text{True}$

iii) Callback when robot leaves the pickup area (pickup cell only) ($\text{robotInPickupQueue} = \text{True}$ and $\text{RobotInPickupArea} = \text{false}$ and $\text{Pickup}(i)$ has more parcels). If the condition is true, call “dequeue pickup queue”, set $\text{robotInPickupQueue} = \text{False}$, set pickup state = open

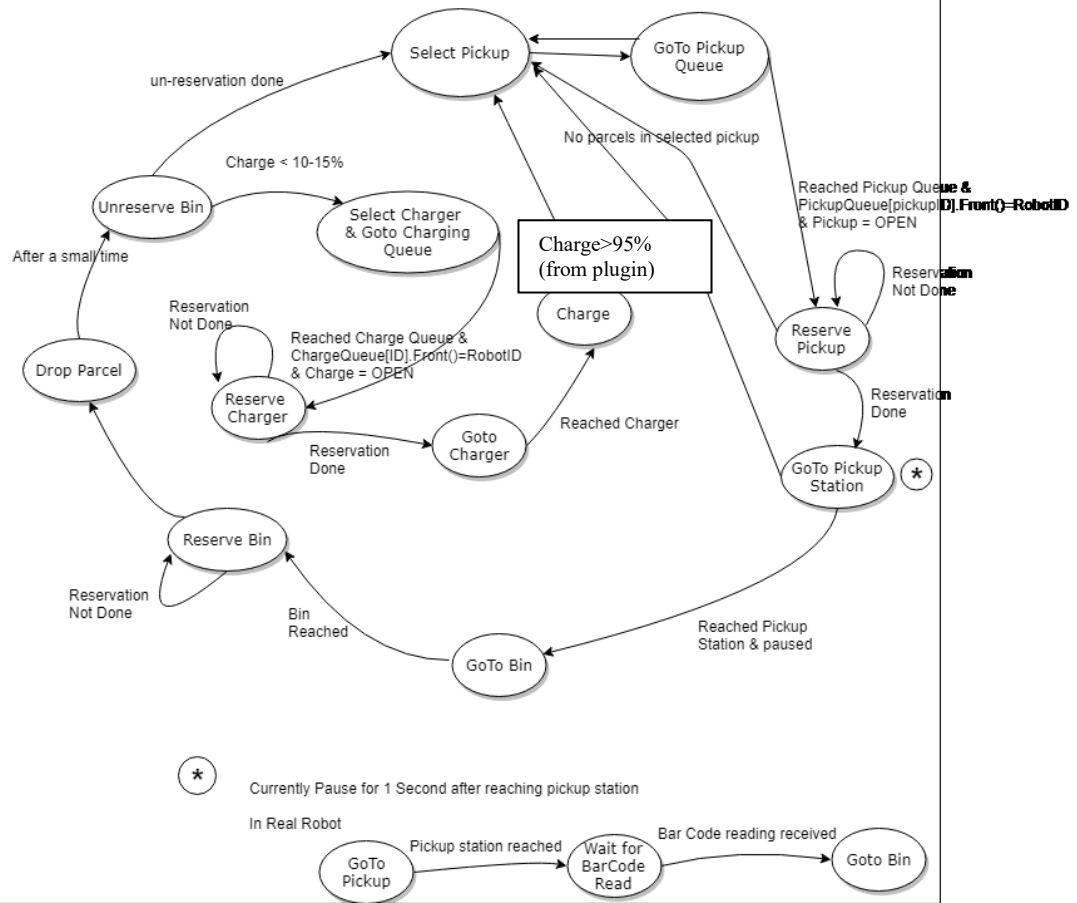
iv) Callback when robot leaves the charge area (charge cell and the next cell) ($\text{robotInChargeQueue} = \text{True}$ and $\text{RobotInChargeArea} = \text{false}$). If the condition is true, call “dequeue charge queue”, set $\text{robotInChargeQueue} = \text{False}$, set charger state = open



v) Callback when the robot is in a pickup queue, but the pickup queue has no parcels. Change state as per BFSM.

In all cases, the information from area is available from the annotated navigation map and the charger/pickup ID selected by the robot

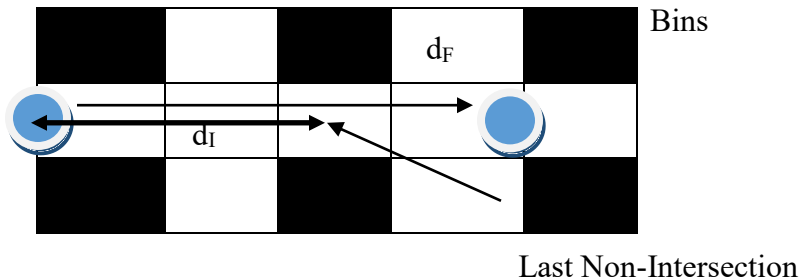
ROBOT TOP LEVEL BFSM



Robot “Goto Goal” Behavior

Level 1a: Goal is forward ahead a certain distance

Assume that the robot senses d_F from the front proximity sensor. The cell corresponding to the physical location of the robot at front can be calculated. The robot must never stop/aim to stop at an intersection. Therefore, before d_F , the first non-intersection road is calculated. Let it be at a distance of d_I . This is only when the behavior is non-turn. Note that, unlike previously, the world coordinate axis system is now in place.



$$v = K_P \min(d(q, \text{goal}), d_I)$$
$$\theta_G = \text{atan2}(G_y - y, G_x - x)$$
$$\omega = K_P \text{angleDifference}(\theta, \theta_G)$$

$d(q, \text{goal})$ = Euclidean distance between current position and sub-sub-goal

d_F = distance from front proximity sensor

d_I = distance to the previous intersection with no robot

Level 1b: Goal is at the same place with a $\pi/2$ orientation

$v = K_P (d(q, \text{goal}))$: iff goal ahead of robot

θ_G = desired goal angle

$\omega = K_P \text{angleDifference}(\theta, \theta_G)$

Level 2a: Traffic Light Aware Motion Planner

Read the current robot position and current a sub-goal directly ahead

Set sub-sub-goal in the path as the last point where traffic light is yellow/red or last point where a turn needs to be performed.

Publish sub-sub-goal

Level 3: Sequencer

Get path from search algorithm

Simply the path as sequences of straight motion and turn only

(e.g. $(1,1), (2,1), (3,1), (4,1), (4,2), (4,3), (4,4)$ becomes $(1,1) (4,1)$ turn $(4,4)$)

while path is not null

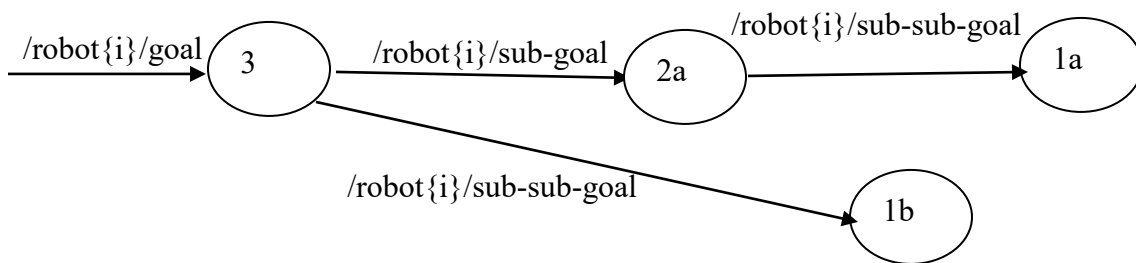
- select sub-goal as next turn/straight path at a maximum distance k

- publish sub-goal and do not wait for action server

- plan a new path from sub-goal

- simply the path as sequences of straight motion and turn only

- wait for action server



RQT-GRAPH: Rqt Graph showing nodes, topics and data to be passed between nodes is shown below:

