86/100

Robot Design Lab
Tutor: Teena Hassan, Mihaela Popescu

WiSe 2022/2023
Group: 09

Joel Köper
Maarten Behn
Nhu Van Nguyen

Worksheet 04

Hand-in date: December 14, 2022

24/25

1 Path Planning

1.1 Quiz on Path Planning and Obstacle Avoidance [25%]

1. **Name 3 path-planning algorithms and describe their pros and cons.** [9%]

9/9

- **Breadth-First Search:**
  Pros:
  - Simple and easy to implement.
  - Suitable for grid with a smaller amount of cells.
  - If there is a available path from start to goal, this algorithm will always find the shortest path between those two nodes.

  Cons:
  - It is not efficient, especially for large grids with a great amount of free cells it will be slow.
  - The Breadth-First Search is a uninformed search, that means it only uses adjacency or neighborhood information and will not optimize the search.
- **Dijkstra's algorithm:**
  Pros:
  - It is a best-first search algorithm (informed search strategy).
  - Is useful when the section between adjacent nodes have non-identical costs.

  Cons:
  - If the actual cost of moving from a node to an neighboring node is equal for all nodes in the graph, then Dijkstra's algorithm is identical to breadth-first search and therefore has in that incident the same cons as Breadth-First Search (as mentioned above).
- **A* algorithm:**
  Pros:
  - It is a well-known and widely used best-first search algorithm for path planning.
  - It is efficient in large grids, because it reduces the search space in comparison to breadth-first search, since it prioritizes the exploration of nodes that have a lower cost.

  Cons:
  - A* is dependent on the accuracy of the heuristic algorithm. They evaluate each node using a cost function (that has two terms: heuristic function and exact cost function). The heuristic function uses a estimated length of the path from the node q to the goal node which also means that the algorithm can possibly overestimate the actual cost of getting to a goal node. It signifies that A* is dependent on the accuracy of the heuristic algorithm.

Further drawbacks:- Additional memory requirement due to the need to store the cost for each frontier node. Th

- A* can be identical to Dijkstra, if the estimation function (heuristic function) is h(x)=0 and will then obviously obtain Dijkstra's disadvantages.

2. **Breadth-first search is optimal, that is, if a path exists, then it always returns the shortest path. Why? [5%])**

5/5

Because the Breadth-First Search is an brute-force search strategy (exhaustive), that means it enumerates systematically all possible nodes and check if any of them is the goal.

Beginning from the start node the breadth-first search checks 1 by 1 if any of the nodes is the goal node and if not it will resume to expand the tests at the adjacent nodes. The breadth-first search skips the previously visited nodes in it's process. The expansion of the tests will continue until the goal is reached or all possible nodes have been visited.

This algorithm searches all paths of length 'n' before checking the paths of length 'n + 1'.

3. **Which condition should be satisfied by the heuristic so that the A* algorithm finds the least-cost path from start to goal? Give two examples of heuristics that satisfy this condition. [3%]**

3/3

- To find the least-cost path from start to goal the heuristic function needs to be admissible. The heuristic function is admissible if it doesn't overestimate the actual cost of reaching the goal node from the current node.

- In a continuous environment for instance outdoor feature maps, the Euclidean distance (length of the shortest part between two points, in a straight line, in an Euclidean space) would satisfy the condition as an admissible heuristic.

- In a discrete environment like an indoor occupancy grid map for example, the Manhattan distance would be an admissible heuristic, because it adds up the total distance travelled along each axis, which makes it fitting for indoor maps.

4. **Why do we need obstacle avoidance? [2%]**

2/2

There are multiple reasons for avoiding obstacles and averting collisions. For example to ensure autonomy, increase robustness of the robot, safety reasons etc. It is worth mentioning that global path planning is to slow for dynamic environments, because it assumes that the positions of obstacles are constant. Furthermore it is commonly the case that the environment is dynamic. For instance a object/obstacle could change location such as another robot moving, a human walking or an item falling from a shelf. Thus, by avoiding collisions, it enables the robot to navigate in an unknown environment.

5. **How are candidate trajectories (or velocities) generated in the dynamic window approach? [6%]**

5/6

The Dynamic Window Approach (DWA) is a velocity control based method and it chooses an optimal velocity for a quite short time interval and it updates this velocity regularly.

At first the dynamic window will be detected. In other terms the set of velocities that the robot can reach during a short time interval of T seconds. Then it will evaluate which of these velocities will not cause an collision with nearby obstacles. The next phase in the DWA is to calculate a score for every admissible velocity in the dynamic window. It will choose the velocity with the highest score and steer the robot with said highest score velocity. The dynamic window approach will continue to repeat these steps until the goal position is reached.

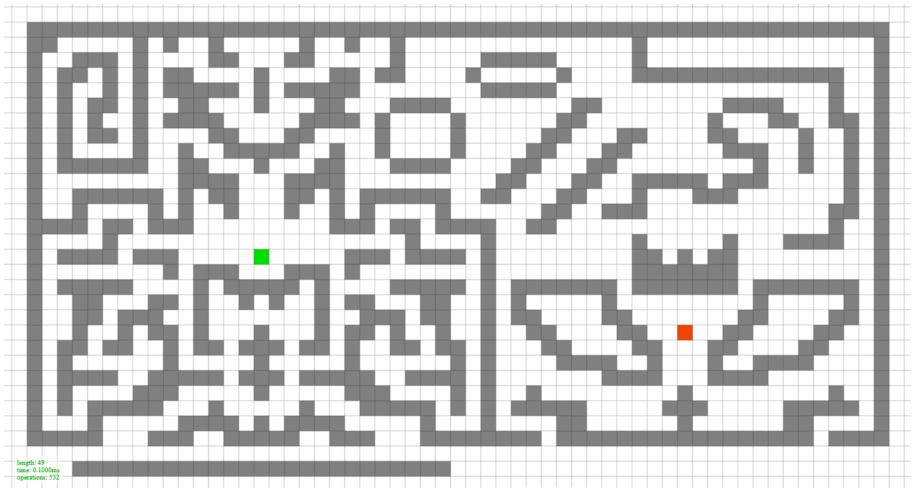
This last part is not necessary to answer this question.Candidate velocities are the admissibl

1.2 Having Fun with Path Planning [10%]

10/10

1. **Create obstacles in the map by clicking on the cells. Select a cell as the start cell (green) and another cell as the goal cell (red). Experiment with the grid design so that you create an interesting map with several obstacles. Submit a screenshot of your grid world.** [2%]

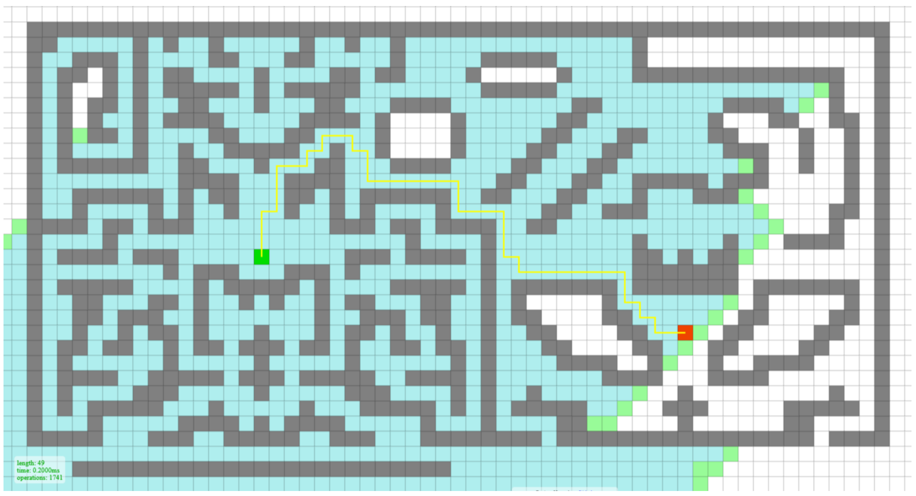
Meticulous design!



```
length: 49
time: 0.1000ms
operations: 532
```

Figure 1: Grid world

2. **Now, use the menu to select Dijkstra's algorithm. De-select the option "Allow Diagonal". Click on "Start Search". Take a screenshot showing the map, the found path, all the visited nodes (light blue) and all the frontier nodes (light green)** [2%]



```
length: 49
time: 0.2000ms
operations: 1741
```

Figure 2: Dijkstra's algorithm

3. **Now select the A* algorithm with Manhattan distance as heuristic. De-select the option "Allow Diagonal". Click on "Restart Search". Take a screenshot showing the map, the found path, all the visited nodes and all the frontier nodes.** [2%]

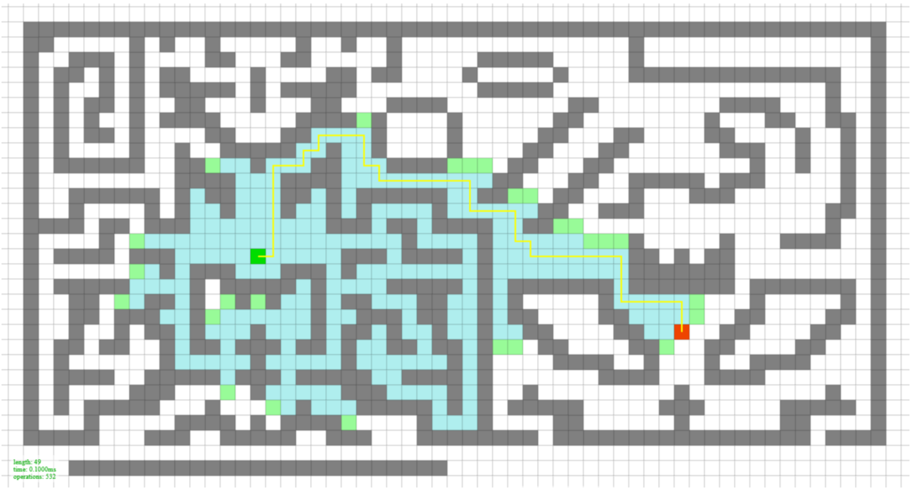


Figure 3: A* algorithm

4. **What differences do you observe in the above results produced by A* and Dijkstra's algorithms? Describe the reasons for these differences.** [4%]

   The Dijkstra's algorithm result has notably more visited nodes in comparison, while A*'s algorithm results display significantly less visited nodes. A* algorithm explored the cells that are closer to the goal node and closer to the start node which reduced the search space. Therefore in this scenario A*'s algorithm is more efficient than Dijkstra's algorithm.

   Nice answer!

   A* is a informed search (greedy best-first approach) and prioritizes the exploration of those nodes that have a lower cost instead of exploring all nodes with equal priority.

   Both algorithms are best-first search algorithms. Yet in this case the result of Dijkstra's algorithm is identical to the breadth-first search, because the cost to move from a node to an adjacent node remains the same in this grid map and Dijkstra's algorithm uses only the exact cost g(q) function, that is the actual cost of moving from the start node to node q. A* on the other hand uses the heuristic function additionally to the exact cost g(q) function. The heuristic function from A* uses a estimated length of the path from the node q to the goal node.

2 Launch Your Nodes! [20%]

2.2 Write the Launch File [20%]

Points = 20%

1. Extend the launch file to start the following 3 nodes. Each node should be started in a separate terminal. [3%]
   I changed the prefix from xterm to alacritty because it is my every-day-terminal and i don't have xterm installed.

   - rdl_move_base node

```
    movebase_node = Node(
                    package="rdl_move_base",
                    executable="rdl_move_base",
                    prefix=['alacritty -e']
                )

    ld.add_action(movebase_node)
```

- rdl_teleop_keyboard node

```
    teleop_keyboard_node = Node(
                package="rdl_teleop_keyboard",
                executable="rdl_teleop_keyboard",
                prefix=['alacritty -e']
            )
    ld.add_action(teleop_keyboard_node)
```

- wheel_velocity_controller node

```
    wheel_controller_node = Node(
                package="worksheet02",
                executable="wheel_velocity_controller",
                prefix=['alacritty -e']
            )
    ld.add_action(wheel_controller_node)
```

The changes for this step are creating the nodes them selves with information about what to run and where to find it as well as attaching the nodes to the launch script with add_action.

2. In the launch file, change the name of wheel velocity controller node to a name of your choice. [1%]

```
    wheel_controller_node = Node(
                package="worksheet02",
                executable="wheel_velocity_controller",
                name="wheel_controller",
                prefix=['alacritty -e']
            )
    ld.add_action(wheel_controller_node)
```

3. In the launch file, change the name of the following topics [4%]:
   - Change turtlebot3_burger/wheel_left_joint_controller/command to rdl_left_wheel_velocity
   - Change turtlebot3_burger/wheel_right_joint_controller/command to rdl_right_wheel_velocity

```
    movebase_node = Node(
                package="rdl_move_base",
                executable="rdl_move_base",
                # Renames topics and services
                remappings=[
                    ("turtlebot3_burger/wheel_left_joint_controller/command", "rdl_left_wheel_velocity"),
                    ("turtlebot3_burger/wheel_right_joint_controller/command", "rdl_right_wheel_velocity"),
                ],
                prefix=['alacritty -e']
            )

    ld.add_action(movebase_node)

    wheel_controller_node = Node(
                package="worksheet02",
                executable="wheel_velocity_controller",
                name="wheel_controller",
                # Renames topics and services
                remappings=[
                    ("turtlebot3_burger/
                       wheel_left_joint_controller/command", "
                       rdl_left_wheel_velocity"),
                    ("turtlebot3_burger/
                       wheel_right_joint_controller/command",
                       "rdl_right_wheel_velocity"),
                ],
                prefix=['alacritty -e']
            )
    ld.add_action(wheel_controller_node)
```

4. Modify the node wheel velocity controller in worksheet02 package so that it takes track_width, wheel_diameter, max_abs_linear_vel and max_abs_angular_vel as parameters [8%].

```
        def __init__(self):
            super().__init__('wheel_velocity_controller')

            self.declare_parameters(
                    namespace='',
                    parameters=[
                        ('track_width', 0.16),
                        ('wheel_diameter', 0.066),
                        ('max_abs_linear_vel', 0.18),
                        ('max_abs_angular_vel', 2.5),
                        ]
                    )
```

Declaration of the parameters them selves with default values.

```
            self.track_width = self.get_parameter('track_width').
                value
            self.wheel_diameter = self.get_parameter('
                wheel_diameter').value


        #This method computes the wheel velocities based on the
            desired linear and angular velocities.
        def calculate_wheel_velocities(self, msg):
            self.left_wheel_velocity = (msg.linear.x - msg.angular
                .z) * (self.track_width / 2) * (2 / self.
                wheel_diameter)
```

```python
        self.right_wheel_velocity = (msg.linear.x + msg.angular.z) * (self.track_width / 2) * (2 / self.wheel_diameter)
```

Assigning the track_width and wheel_diameter parameters to variables and calculating the cmd_vel values with them.

```python
        if msg.linear.x <= self.get_parameter('max_abs_linear_vel').value and msg.angular.z <= self.get_parameter('max_abs_angular_vel').value:
            self.calculate_wheel_velocities(msg)
            self.publish_wheel_velocities()
        else:
            self.get_logger().info('Too high linear or angular velocity! Please reduce the velocity')
```

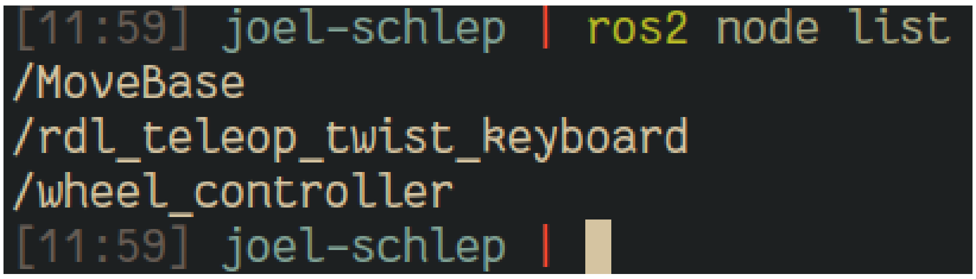
Replacing the hard-coded velocity bounds with parameters.

5. After successfully building your modified wheel_velocity_controller, set its parameters wheel_base, wheel_diameter, max_abs_linear_vel and max_abs_angular_vel to their correct values through the launch file. [4%]

```python
    wheel_controller_node = Node(
                package="worksheet02",
                executable="wheel_velocity_controller",
                name="wheel_controller",
                # Renames topics and services
                remappings=[
                    ("turtlebot3_burger/wheel_left_joint_controller/command", "rdl_left_wheel_velocity"),
                    ("turtlebot3_burger/wheel_right_joint_controller/command", "rdl_right_wheel_velocity"),
                ],
                parameters=[{"track_width": 0.16}, {"wheel_diameter": 0.066}
                            , {"max_abs_linear_vel": 0.18}, {"max_abs_angular_vel": 2.5}],
                prefix=['alacritty -e']
            )
    ld.add_action(wheel_controller_node)
```

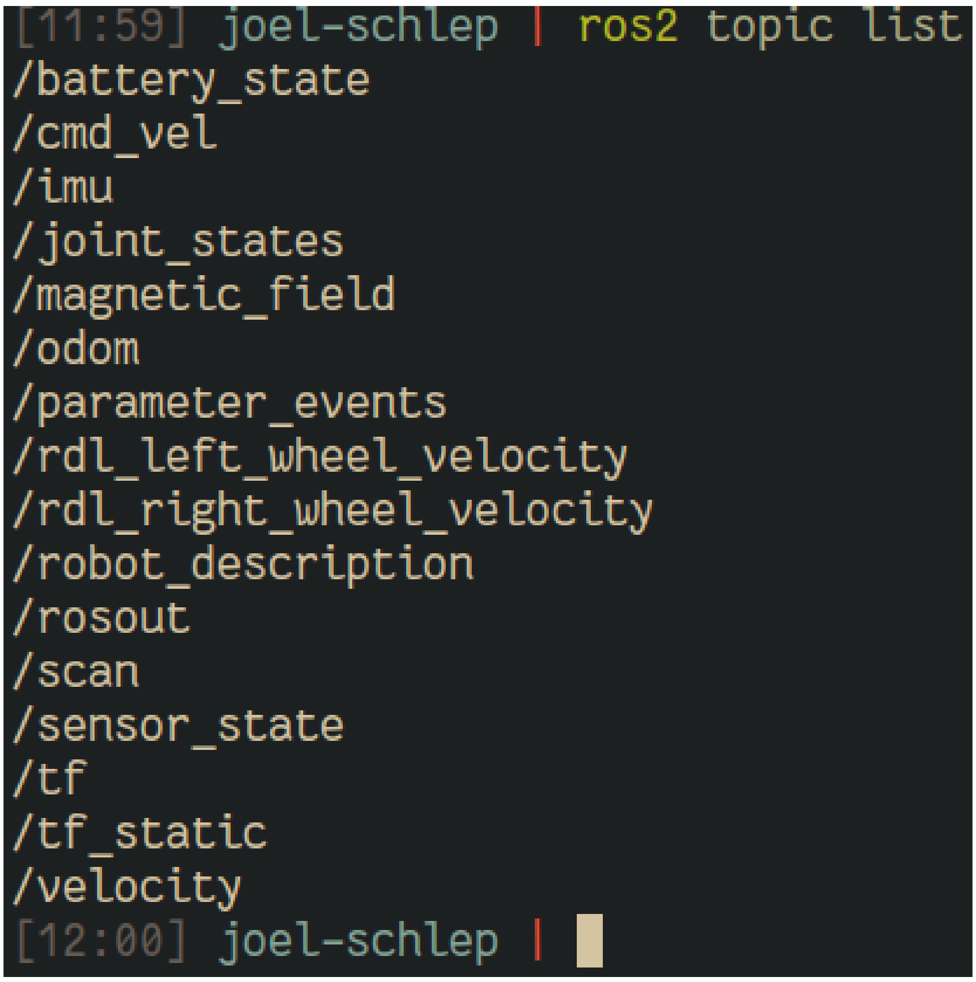
2.3 Build and Launch

- ros2 node list[4]
- ros2 topic list[5]
- ros2 param list[6]
- ros2 param dump wheel_controller[7]



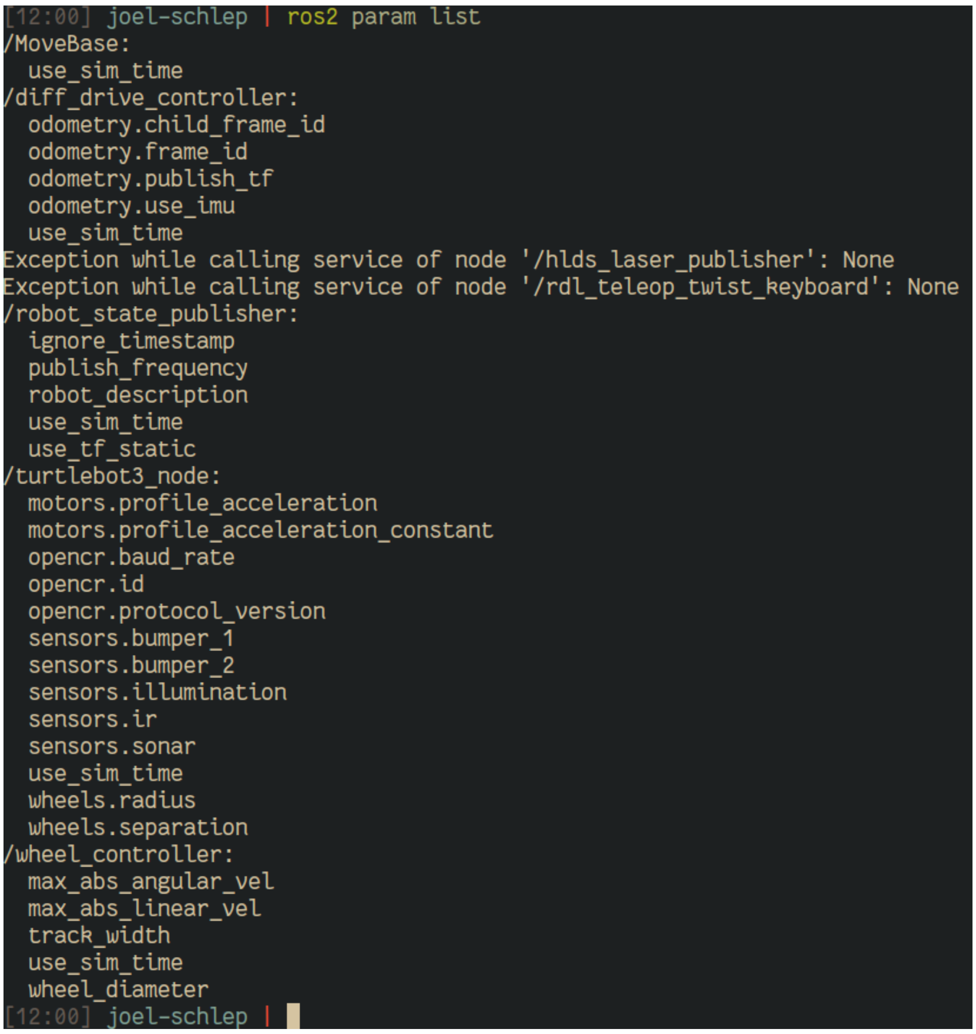
```
[11:59] joel-schlep | ros2 node list
/MoveBase
/rdl_teleop_twist_keyboard
/wheel_controller
[11:59] joel-schlep |
```

Figure 4: ros2 node list



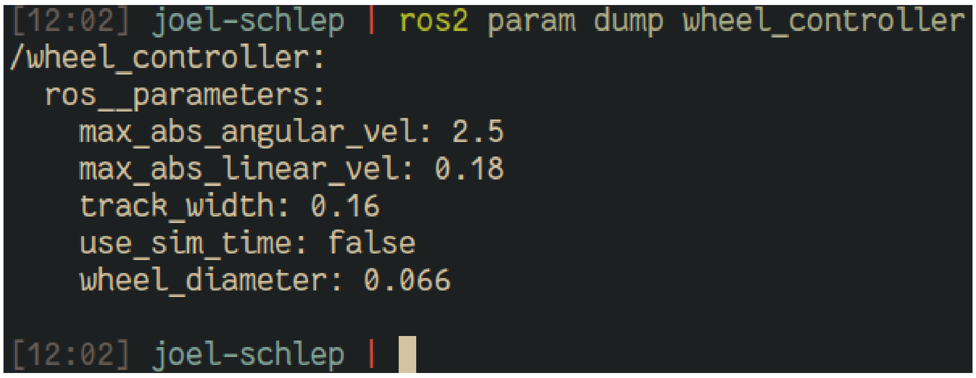
```
[11:59] joel-schlep | ros2 topic list
/battery_state
/cmd_vel
/imu
/joint_states
/magnetic_field
/odom
/parameter_events
/rdl_left_wheel_velocity
/rdl_right_wheel_velocity
/robot_description
/rosout
/scan
/sensor_state
/tf
/tf_static
/velocity
[12:00] joel-schlep |
```

Figure 5: ros2 topic list



```
[12:00] joel-schlep | ros2 param list
/MoveBase:
  use_sim_time
/diff_drive_controller:
  odometry.child_frame_id
  odometry.frame_id
  odometry.publish_tf
  odometry.use_imu
  use_sim_time
Exception while calling service of node '/hlds_laser_publisher': None
Exception while calling service of node '/rdl_teleop_twist_keyboard': None
/robot_state_publisher:
  ignore_timestamp
  publish_frequency
  robot_description
  use_sim_time
  use_tf_static
/turtlebot3_node:
  motors.profile_acceleration
  motors.profile_acceleration_constant
  opencr.baud_rate
  opencr.id
  opencr.protocol_version
  sensors.bumper_1
  sensors.bumper_2
  sensors.illumination
  sensors.ir
  sensors.sonar
  use_sim_time
  wheels.radius
  wheels.separation
/wheel_controller:
  max_abs_angular_vel
  max_abs_linear_vel
  track_width
  use_sim_time
  wheel_diameter
[12:00] joel-schlep |
```

Figure 6: ros2 param list



```
[12:02] joel-schlep | ros2 param dump wheel_controller
/wheel_controller:
  ros__parameters:
    max_abs_angular_vel: 2.5
    max_abs_linear_vel: 0.18
    track_width: 0.16
    use_sim_time: false
    wheel_diameter: 0.066

[12:02] joel-schlep |
```

Figure 7: ros2 param dump

3 Task Planning

22/25

3.1 Task Planning and Acting (25%)

Please answer the following questions:

1. Describe briefly any four assumptions of classical task planning. (4%)

   - A0: Finite system:
     Meaning that every Object in the world has a finite number of possible states and actions it can perform. Therefore the entire system has possible states and actions.
   - A1: Fully observable: The agent always knows the current state of the world.
     The agent can fully observe everything in the world. There are no unknown states, therefore the agent knows the current state of the world.
   - A2: Deterministic:
     The Each action in the world has only one outcome. And if the same action is performed in two world with the same state, the new state of the two worlds is also the same.
   - A3: Static (no exogenous events):
     Every state change in world is caused by an action the agent performs. There are no other actors, that can change anything in the world. For example things like people running through the area a robot operates in is defined to be not possible.

2. What is the difference between an operator and an action? (1%)

   Operators are generalizations of actions. A Operator has a name, a set preconditions and a set of effects. For expample is pickup the general operator to pickup any object. An action is a ground instance of an operator. For example the operator pickup with specific target is an action. An operator can not be applied to a world, because it is only a template for an action. Only if one specifies the generalized information of on operator it becomes an action an can be appield to the world. Very nice!

3. Distinguish between planning domain and the statement of a planning problem (4%).

   -3

   A planning domain is the world in which the planning problem is defined. It represents all possible states the system can have. For example in path planning the planning domain is the map. And the map represents the set of all possible positions and rotations a robot can have. The planning domain include all the constant symbols, predicates and operators, that define the entities in the e

   The statements of a planning problem, is the set of actions that performed achive the goal state. Given a planning domain, we can define a planning problem. The statement of the planning problem includes the initial state, a set of operators that can be used and the

4. How would you determine if an action ‘a’ is applicable in a state ‘s’? (1%)

   If s satisfies all the positive preconditions of a and has non of the negative preconditions of a.

5. How would you determine if a state ‘s’ satisfies the goal ‘g’? (1%)

   A state s satisfies the goal state g if s has alle the posivite goal conditions and non odf the negative goal conditions.

6. What do you understand by the term ‘state transition’?(2%)

   A state transition is the change from one state to an other state. The would probably not be the same. The effects of the action that performed the state transition can be defined as the delta between state one and two.

fly is the operator symbol.plane is a paramet

7. Given below is an operator fly(plane, from, to). fly-plane is the operator symbol and plane, from and to are parameters of the operator. The planning domain contains an Airbus A320 and a list of domestic airports in Germany. Construct the action to fly A320 from Hamburg (HAM) to Frankfurt (FRA). (6%)

```
fly(Airbus A320, HAM, FRA)
    precond: can-fly(Airbus A320), is-airport(HAM), is-airport(FRA),
        at(Airbus A320, HAM)
    effects: !at(Airbus A320, HAM), at(Airbus A320, FRA)
```

8. Represent the above action as a behaviour tree by applying the postcondition-precondition-action rule. (4%) (Note: For simplicity, you can replace the negation symbol ‘!’ with the prefix ‘not-’, e.g. ‘not-at(locationA)’, and use this as the label for the corresponding condition node. Remember that you should use grounded predicates and action names for labeling the leaf nodes in the tree.)

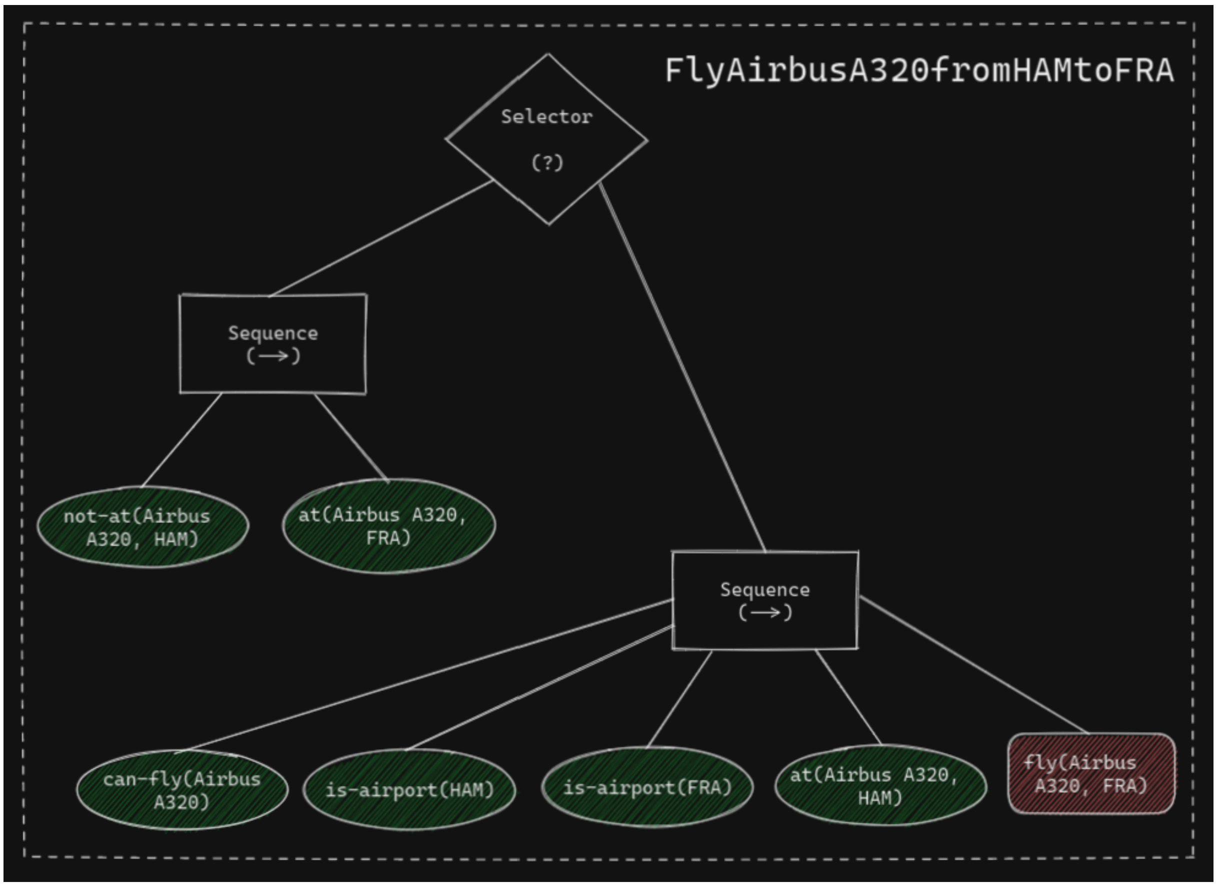


Figure 8: Behaviour tree FlyAirbus320fromHAMtoFRA

9. Describe briefly the control logic of the above behaviour tree. (2%)

   The behaviour tree FlyAirbus320fromHAMtoFRA implements the control logic of fly(Airbus 320, HAM, FRA). The root node of the behaviour tree is a selector node. This selector node has two children. The first represents the case where the action does not need to be performed, because all the post-conditions of the action are already true. The other case represents the case where the action will be performed if the all the precondition's are true. First the tree would step into the root node an then into the first child of the selector. This is a Sequence node. Then all the children conditions of the Sequence node will be checked till one is false and return failed. The the sequence node would return failed. If all the children node are true and therefore return Successful the sequence node returns Successful. If the Sequence node returns Successful the root Selector node will return Successful. If the Sequence node returns Failed the Programm will step into the second child of the root node. The program will then again step into every child node of the Selector till one conditions is false and returns failed. If all the pre condition's return Successful the programm will step into and run the Action. If the action returns Successful the entire tree will return Successful. Otherwise it will return Failed.

3.2 Applying Task Planning to Deliver Medicines (20%)

In this video https://www.youtube.com/watch?v=AWtIybYU-20, you will see an application of autonomous drones to deliver medicines to remote places. After watching the video, solve the following subtasks.

3.2.1 Planning Domain (10%)

10/10

Here, you will define a simple planning domain for the above application scenario. For this, do the following:

1. Define constant symbols to represent a package, the drone, the base station of the drone, and the location of the hospital. (2%)

   (a) Drone: d

   (b) Package: p

   (c) Base Station: b

   (d) Hospital Location: h

2. Define predicates representing where the drone is located and whether a package is attached to the drone or has been released from the drone. (2%)

   (a) at(x, y) - Where x is the object and y is the Location

   (b) on-drone(x, y) - Where x is the object and y is the drone it is attachet to.

   (c) is-released(x) - Where x is a package.

   (d) in-air(x) - Where x is a drone.

3. Define operators for the drone to take-off, land, hover, fly and release package. If your operators need additional predicates, add them to the list of predicates that you defined in the previous subtask. (6%)

```
take-off(d, p)
    precond: at(d, b), on-drone(p, d), !in-air(d), on-drone(p), !is-released(p)
    effects: in-air(d)

land(d)
    precond: in-air(d), at(d, b)
    effects: !in-air(d)

hover(d)
    precond: in-air(d)
    effects:

fly-to-hospital(d)
    precond: in-air(d), at(d, b), !at(d, h)
    effects: at(d, h), !at(d, b)

      base-station
fly-to-hospital(d)
    precond: in-air(d), at(d, h), !at(d, b)
    effects: at(d, b), !at(d, h)

                                                  on-drone(p,d)
release (d, p)
    precond: in-air(d), at(d, h), !at(d, b), on-drone(p), !is-released(p)
    effects: !on-drone(p), is-released(p)
```

3.2.2: Not attempted0/