Belief and World Representations in Duckietown

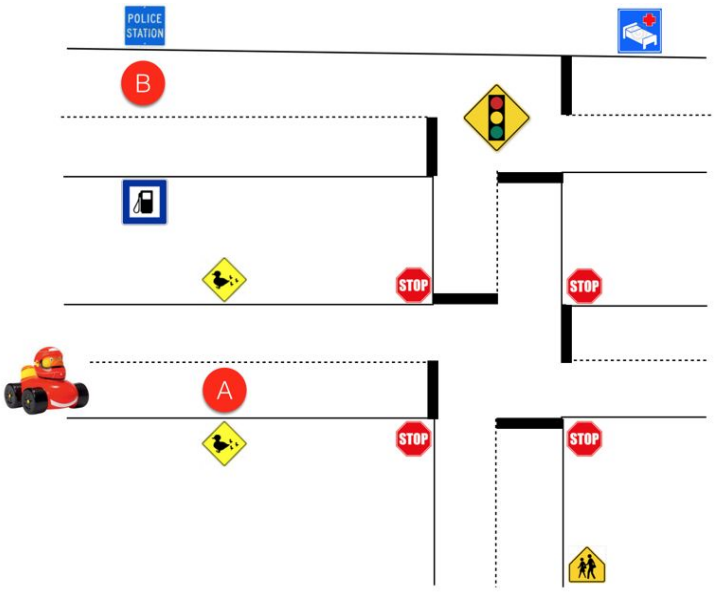
What is a *representation*? (i.e., how the robot models the external world?)
A representation is a symbolic structure that encodes the knowledge of the robot about the external world as well as its state. This symbolic structure is implemented in a data structure that can be understood/used by perception and planning algorithms. Clearly, a possible choice of representation is the set of all sensor measurements collected by the robot: these measurements provide a "picture" of the internal and external state of the world. However, such a representation is not very efficient: its size grows over time (as new measurements are collected) and it may be computationally demanding to access and perform computation over such a large amount of data. This motivates the interest towards more compact representations. Ideally, a *minimal* representation stores the smallest amount of data that is sufficient for the robot/car to perform a given task. More concretely, we will think about a representation as a set of parameters that encodes the world's and robot's state, and we will prefer representations using small number of parameters.

Involved modules:

- SLAM
- global localization (user)
- coordination
- navigation/path planning

What is the task?
Pickup duckie at point A and drop it at point B following the shortest path, avoiding collisions with other vehicles, and respecting traffic rules.

Working example:

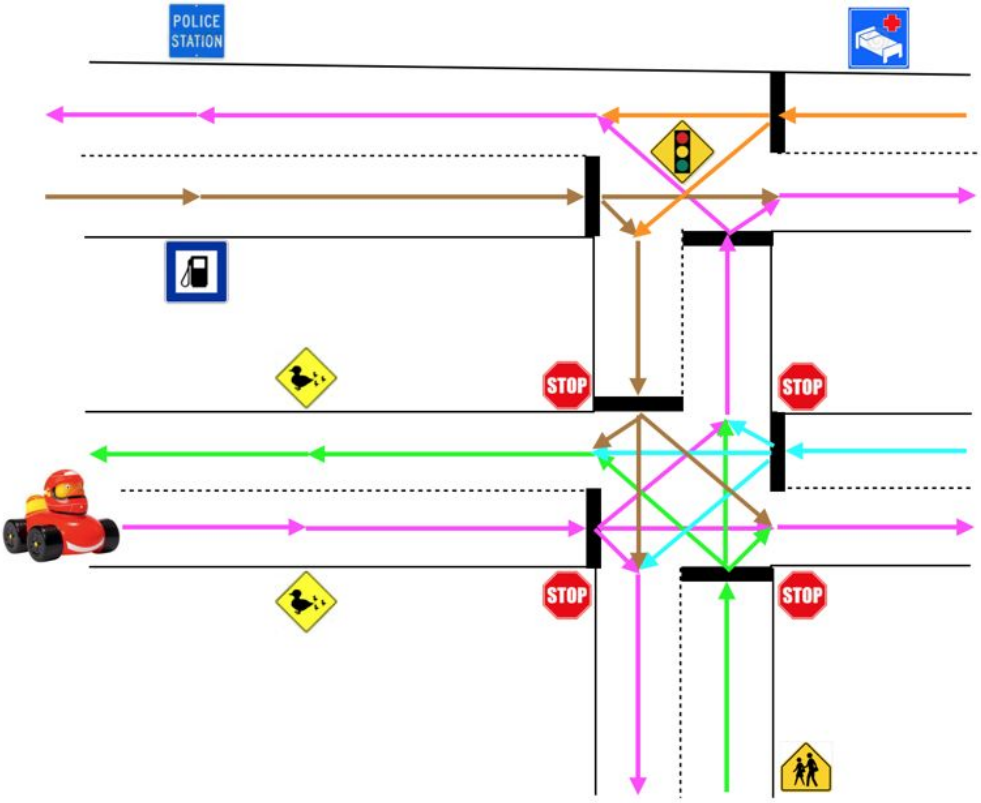


Disclaimer ("From ~~Bat~~Duckman: the ~~Dark~~Duckie Knight"):
"It is not the representation that Duckietown *needs*, but the one that it *deserves* right now. So we'll implement it. Because it can take it."

Meaning: the described representation is not necessarily minimal for any given task. However, it can be projected down to produce a minimal representation for a specific task (e.g., localization and mapping). In this sense it can be considered minimal for a *set of tasks* and for our definition of these tasks.

1. World representation (a.k.a. map representation):

We represent the world as a directed graph, where **nodes** corresponds to geographic locations and **edges** represent connectivity among these places (i.e., if there is an edge between i and j, we can drive directly between the two locations). Both edges and nodes have **attributes**, which are useful to support decision making and control. The directed graph for our working example is:



where "arrows" denote directed edges, while we imply that at both ends of each arrow there is a node in the graph. Edges are shown in colors only for visualization purposes.

1.1 Node attributes

Each node corresponds to a place we would like to "remember". Examples of those places are: intersections, possible duckie pickup stations, possible destinations. For each place we store the following attributes:

- position: (x,y)
- (optional) position uncertainty: 2 by 2 information matrix
- types of nodes:
    - intersection with stop (stop might be an external landmark)
    - intersection with traffic light (traffic light might be an external landmark)
    - OUT the intersection: desired by Javier to understand "shared" lanes for multi robot coordination (project)
    - label: intersection nodes (between intersection IN and intersection OUT)
- (optional) "places" pickup / dropoff
- (optional) name: {school, parking lot, hospital, police station, gas station, intersection Daffy St./Duck St., ...} The name is not necessary, but might help visualization and debugging. Also it enables other functionalities, e.g., the duckie sets "school" as a destination.

Questioning the assumptions:

- why nodes with positions?
    - duckies request a pickup with a GPS location, hence the car needs to know the GPS location (x,y) of each place
- why nodes with uncertainty?
    - strictly speaking, the uncertainty is not necessary. However, if there are regions for which the quality of the map is poor, the uncertainty information may help planning smarter routes.
- why nodes with type?
    - the knowledge of the node type helps satisfying the traffic rules. For instance, it is cool to traverse a "duckie pickup" node at 35mph -if there is no duckie to pick up- but it is not cool to traverse an "intersection with stop" node at 35mph.

1.2 Edge attributes
Directed edge = lane, with one direction
There might be multiple lanes that are adjacent

Each edge defines the traversability between two places and can be understood as lane segments which separate two places/nodes. Each edge has the following attributes:

- id: unique integer from 1 to m (total number of edges)
- type: {straightLane, intersectionRightTurn, intersectionLeftTurn, intersectionStraight, ...}
- length: used as a weighting; distance to cover to move between the adjacent nodes (this may be different from the Euclidean distance for curved paths)
- width: distance between the lane markings defining the lane segment
- (optional) type_left_marking:
- (optional) type_right_marking:
- (optional) speed limit
- (optional) name: {"Duckie St."}
- (optional) right_neighbor: None | ID
- (optional) left_neightbor: None | ID
- extra appearance cues:
    - a list of ("landmark", edge, distance along edge), which can be mainly used to store the position of traffic signs

Questioning the assumptions:

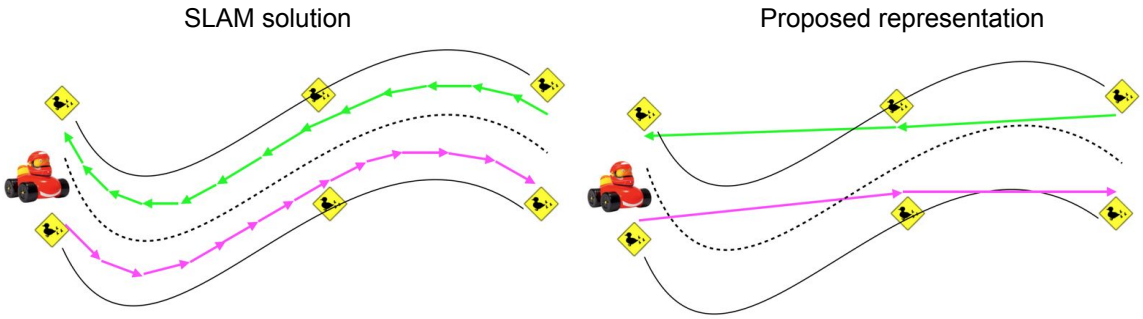
- why edges with id?
    - later we'll need the id to understand "where" the car is on the map
- why edges with length?
    - this is useful for the planner, in order to find shorter routes
- why edges with width?
    - this helps the lane controller to keep the car in the middle of the lane. For instance, if the lane controller sees a single lane marking line, it cannot establish the center of the lane hence it cannot stabilize the car in the middle.
- why edges with right./left_neighbor?
    - the planner needs to know which edges are contiguous to plan lane changes. After performing SLAM (see later), we lose this metric information, hence we need to store this topological flags

1.3 How do we create this representation?

From sensor data:

1. Run your favourite SLAM algorithms (pose graph optimization) on sensor data. Include as pose in the graph all car poses -sampled at regular time/space intervals- plus the position of external signs and traffic lights. We call the two of nodes "car poses" and "landmarks". After optimizing the pose graph:
    a. For each node, assign (from the SLAM solution)
        i. position and (marginal) covariance
    b. For each edge, assign (from the SLAM solution):
        i. length (Euclidean distance between adjacent nodes, assumed close)
        ii. right_neighbor and left_neightbor: looking at nodes metrically close to the extremes of the current edge
2. Independently from pose graph optimization:
    a. For each node, assign (using sensor data)
        i. type (possibly "unknown")
    b. For each edge, assign :
        i. unique id
        ii. width (using sensor data, possibly averaged over the last n measurements)
        iii. type_left_marking, type_right_marking
3. eliminate intermediate nodes by "merging" edges that share an unnecessary node
    a. find the set of "places" node: a place node is the closest car pose to a given landmark
    b. for each "non-place" node B, merge the adjacent edges (A,B) and (B,C) as follows:
        i. substitute edges (A,B) and (B,C) in the graph with (A,C)
        ii. for other landmarks, save:
            1. landmark type and distance from extreme from adjacent node
        iii. assign the following attributes to (A,C):
            1. length(A,C) = length(A,B) + length(B,C)
            2. width(A,C) = ½ (width(A,B) + width(B,C)) (average)
            3. distance(landmark) should be also augmented accordingly
        iv. renumber edges from 1 to m

Note: while the edges in the pose graph solution are practically a metric representation of the lane, this information is lost when decimating the nodes as described in Section 1.3. For instance, compare the following cases:



The information attached to the edges of the graph on the right is not sufficient for low-level control (lane following), which we entrust to the reactive control of the lane_controller

Using the map editor:

- to be written

2. Map representation for localization

We use Markov localization (ML) to estimate where the car is on the road network. ML estimates the probability of the car being at a given, discrete state.

We perform ML on a graph, where each node represents a 2D position and each edge represents a possible transition between 2 nodes. ML estimates a probability for every node in the graph, and we denote with P(i,t) the probability of the car being in node i at time t, given all measurements collected till time t. We call this probability distribution the *belief* of the car. More formally, the **position belief at time t** is the conditional probability of the position of the car at time t, given all the past measurements.

One could perform Markov localization directly on the graph described in Section 1. However, such "map" is very coarse (nodes = "places") and we would like to have a finer description of where the car is. This way, for instance, we are able to provide information as "The next intersection is in 50m". For this purpose, we introduce extra nodes in the graph, by splitting the edges at regular intervals

Remark: rather than "super-sampling" extra nodes, we can keep the nodes produced by the SLAM algorithm.

Remark: *"Do not try and add nodes. That's impossible. Instead... only try to realize the truth: there are no nodes."*

The following ingredients are what's needed for ML:

- **State space**: as mentioned before the state space of the ML is a set of nodes, each one having a unique id.
- **Transition probabilities**: each edge in the graph has associated a nonzero transition probability. Note that the edges are not the edges in the SLAM graph (which represent a single realization of the transition probabilities). Hence as a postprocessing of the graph, one should create the edges corresponding to all the possible transitions. Transition probabilities depend on the following measurable quantities:
    - speed (1D odometry) in the lane direction
    - lane_change: binary detections of lane changes with corresponding (left/right) direction
- **Measurement model**: we measure the following quantities:
    - 2D position of external landmarks at known locations.

The following quantity can be estimated independently from the ML (e.g., using a Kalman filter):

- **y**: 1D position with respect to the lane
- **phi**: angle with respect to the lane direction

Note: while the ML uses the odometry and the sign detection, the measurements used to estimate dw and phi come from a different sensor modality (lane detection). For this reason we can safely decouple the estimation of these quantities.

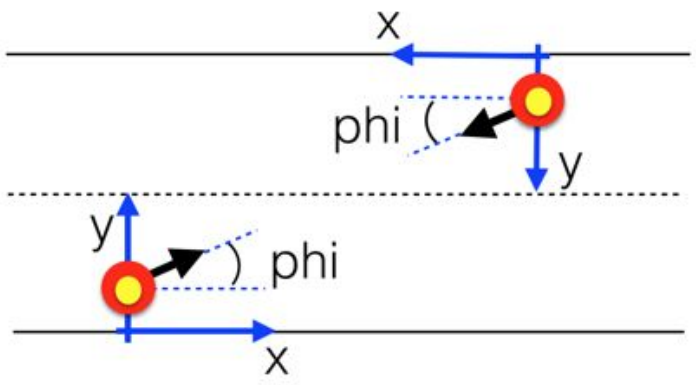
Reference frames: to make the representation unambiguous we should specify what is the reference frame of y and phi. The reference frame for each duckie is defined as follows, for a straight lane segment:

- has the x axis parallel to the lane direction and aligned with the allowed direction of travel
- it has the origin attached to the right lane marking and at a position corresponding to the projection of the duckie position along the lane direction (it is a moving frame that moves with the duckie)

Note that the reference frame for y and phi is moving with the car. Moreover, by definition of the reference frame, the x-coordinate of the duckie is always zero, while:

- the y coordinate represents the distance of the center of the car from the closest right lane marking
- the phi angle is positive for counter-clockwise rotations

Moreover note that this reference frame is local to each duckie. The following figure shows as example of reference frames attached to two different duckies.



origin should be the middle of the lane

Map representation produced by SLAM

- nodes at “red” stop lines, and should associate traffic light to those nodes

Map representation for global localization

- graph with sparser set of nodes
- Localization: probability of being at each node
  - if largest probability peak is large: we publish current position + flag: “localized”
  - if largest probability peak is NOT large: we publish n most likely positions + flag: “uncertain”

Map representation for navigation

- path planning: Dstar on graph in which each edge has length information
- navigation: edge should also tell if it’s a right/left turn/straight
- what about start and end nodes?

Map representation for coordination

- module: LED-based (no specs)
  - detection of other vehicles (cannot detect moving vehicles at intersection)
  - binary detection at each of the incoming streets (do not care about shape of intersection)
- project:
  - localization results may be useful at (1-5cm)
  - Desired feature: shape of intersection (T, cross)

Map editor

- node in map editor should contain label for each node about being entering of intersection
- TILES ARE REALLY ONLY NEEDED FOR VISUALIZATION AND DEBUGGING, BUT NICE TO HAVE

Simulator

- disconnected from map representation

Potential project:

- overtaking
- detection of vehicle to pass
- detection of coming vehicles from opposite direction
- lane width