Frames

GPS → map : ? - originPos + size(map)/2

map → GPS : ? + originPos - size(map)/2

yaubot → GPS : rotationMatrix x ? + yaubotPos

GPS → yaubot : rotationMatrix^{-1} x (? - yaubotPos)

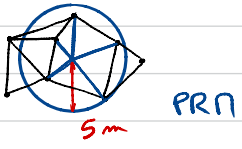
Init

X Y cells/m

- map = OccupancyGrid(50, 50, 5) : non binary / based on probabilities
  - → $0 \leq p < 0.5$: free
  - → $0 = 0.5$: unknown
  - → $0.5 < p \leq 1$: occupied
    - ↳ inflate(map) increase obstacles size

controller = PurePursuit (GPS frame)
- ↳ LinearVelocity = 1 m/s
- ↳ MaxAngularVelocity = 1 rad/s
- ↳ LookaheadDistance = 1 m

prm = PRM (map frame)
- ↳ NumNodes = 50
- ↳ ConnectionDistance = 5m
- ↳ map = inflate(copy(map), yourbot_radius)
  - => increase obstacles size of yourbot radius



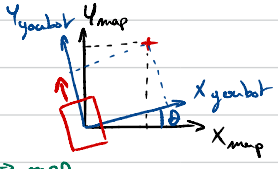
Mapping

- Map is considered of unknown size and our original position in the whole map is considered unknown.
  - => large enough map ⊕ original position in the middle of our map
  - => w/ GetObjectPosition : get GPS original value to have a reference
    w/ " Orientation : get orientation " "
  - => original pose is placed in center of map
  - => $pose_{map}$ = GetObjectPosition − originalPos + size(map)/2

Dpose with reference

- Hokuyo sensor => ray end points ⊕ points in contact with an obstacle

  - 3xN matrix expressed in youbot frame
    => need to express in map frame coordinates



$$\left.\begin{aligned} X_{map} &= X_{youbot}\cdot\cos\theta - Y_{youbot}\sin\theta \\ Y_{map} &= Y_{youbot}\cdot\cos\theta + X_{youbot}\sin\theta \end{aligned}\right\} + youbotPos - originPos + size(map)/2$$

youbot -> GPS -> map

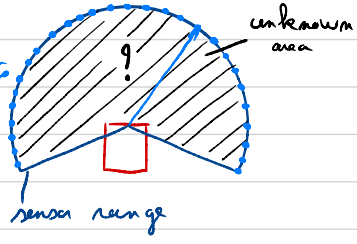
$youbotPos_{map}$

non rotated youbot frame

Dpose

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix}_{map} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}_{youbot} + youbotPos - originPos + size(map)/2$$

rotation matrix

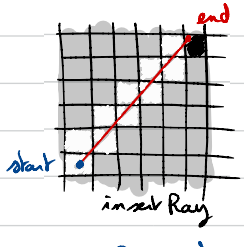


  - only ray end points: unknown points between
    => OccupancyGrid.insertRay
    ⊕ " .updateOccupancy

insertRay(map, $youbotPos_{map}$, $rayEndpoints_{map}$, values)

values = [0.3 0.5] : mid points and end points probabilities

0.5 because end points not necessarily obstacles



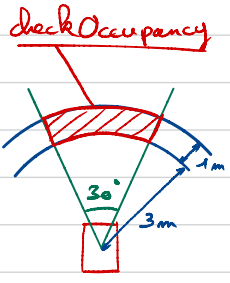
=> updateOccupancy(map, [rotationMatrix * pts(1:2, contacts)]' + $youbotPos_{map}$)

x,y logical matrix

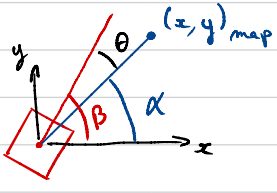
$rayEndpoints_{map}$ downsampled to increase performances
less endpoints is not problematic because the map is rough
compensate number of points w/ large number of map updates

Navigation

- inflatedMap = copy(map) then inflate(map, robotRadius) to avoid obstacles during navigation



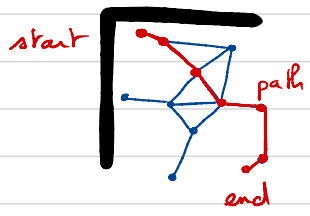
- Choose point of interest to navigate to
  - → meshgrid around youbot
  - → not too far nor too close ($3 < d < 4$)
    $R = hypot(x, y)$
  - → in front of youbot (30°)
    $\theta = atan2(x / -y) - youbotEuler_{\vec{j}}$
    $x / -y$ because VREP 0° is $/-\vec{y}$
  - → handle already visited places } how? not implemented yet
  - → privilegiate corners



check conditions on each combinations of x and y
- => convert $(x, y)_{map}$ coordinates to grid coordinates to remove duplicates
- => checkOccupancy(map, cells, 'grid')
- => get only free cells coordinates
- => convert to map (x,y) then to GPS: grid2world(map_{inf}, targetGridCell)
  targetGridCell random free cell, no algorithm to choose particular path

① If no point found: can't increase max distance because of sensor max range. So increase start angle by 10° each time

- Update Probality RoadMap
  - → pass inflated map to PRM ⇒ [start … end] (pa)
  - → update nodes
  - → findpath



① If no path found, add to nodes and update PRM

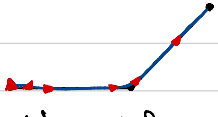
- → get path coordinates in GPS frame
  path = path − size(map)/2 + originPos map → GPS
- → set path to controller
- → push each points in FIFO list to get current intermediate target

Driving

- If we have path registered and encounter goal while driving: repath
- Each time we reach an intermediate point (radius 0.2 ~ 0.3)
  - → if FIFO list is nearly empty: clear and repath
  - → else, pop top and set current target as front object

To drive, we use PurePursuit controller which provides linear and angular velocity when passing current position and orientation

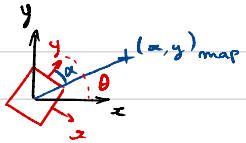

Base behavior


Wanted behavior

Yourbot has omnidirectional wheels so it can move sideways
↳ compose speed
⇒ get angle between front axis and target
↳ get target position in yourbot frame

GPS → youbot

$$(x,y)_{youbot} = rotationMatrix^{-1}\left[(x,y)_{GPS} - youbotPos_{GPS}\right]$$

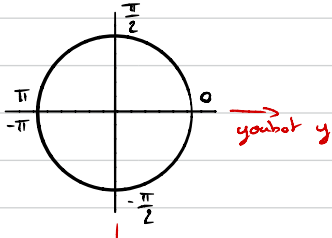


$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix}_{youbot} = \underbrace{\begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{rotationMatrix^{-1}} \left[\begin{pmatrix} X \\ Y \\ Z \end{pmatrix}_{GPS} - youbotPos_{GPS} \right]$$

$$(X\ Y\ Z)_{youbot} = \left[(X\ Y\ Z)_{GPS} - youbotPos_{GPS}\right] \underbrace{\begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{(rotationMatrix')^{-1}}$$

front and back inverted

$\Rightarrow \theta' = \theta + \pi \Rightarrow pos'_{youbot} = -pos_{youbot}$



$\Rightarrow$ angle between youbot front axis and target

$\alpha = atan2(-x, y)$

$\Rightarrow$ compute target rotation matrix

$$targetRMatrix = \begin{bmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{bmatrix}$$

} because multiply like matrix

robot velocity: $[v\ 0] \rightarrow [v.\cos\alpha \quad v.\sin\alpha]$

$\Rightarrow$ youbot_drive($\ominus$ robotVel(1), robotVel(2), rotateVel)

↓ because youbot motors are reversed