Robot Motion

14 October 2018 18:44

Degrees of Motion Freedom (DOF):

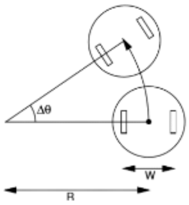
- 1D - 1 DOF
- 2D - 3 DOF - 2 translational, 1 rotational
- 3D - 6 DOF - 3 translational, 3 rotational

Holonomic - move instantaneously in any space of its DOF

Differential Drive:

- Two motors, one per wheel
  - Equal speed for straight-line
  - Equal and opposite direction for turn on the spot
  - Other combinations - circular motion

Circular Path:



$v = velocity\ of\ wheel\ on\ ground$
$r = wheel\ radius$
$\omega = angular\ velocity\ of\ wheel$

$v = r\omega$

Straight line: $v_L = v_R$
Turn on spot: $v_L = -v_R$

Period of motion $\Delta t\ through\ angle\ \Delta\theta$:

Left wheel distance moved: $v_L \Delta t$
Left wheel radius of arc: $R - \frac{W}{2}$
Right wheel distance moved: $v_R \Delta t$
Right wheel radius of arc: $R + \frac{W}{2}$

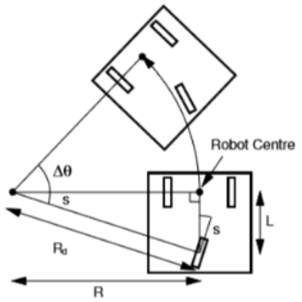
Both wheels subtend same angle so:

$$\Delta\theta = \frac{v_L \Delta t}{R - \frac{W}{2}} = \frac{v_R \Delta t}{R + \frac{W}{2}}$$

Finally:

$R = \frac{W(v_R + v_L)}{2(v_R - v_L)}$ and $\Delta\theta = \frac{(v_R - v_L)\Delta t}{W}$

Circular Path of a Car-Like Tricycle:



- $R = \frac{L}{\tan s}$
- $R_d = \frac{L}{\sin s}$
- $\Delta\theta = \frac{v\Delta t}{R_d} = \frac{v\Delta t \sin s}{L}$

Motion in 2D:

- Straight line of distance D
  - $\begin{pmatrix} x_{new} \\ y_{new} \\ \theta_{new} \end{pmatrix} = \begin{pmatrix} x + D\cos\theta \\ y + D\sin\theta \\ \theta \end{pmatrix}$
- Pure rotation of angle α
  - $\begin{pmatrix} x_{new} \\ y_{new} \\ \theta_{new} \end{pmatrix} = \begin{pmatrix} x \\ y \\ \theta + \alpha \end{pmatrix}$
- Circular Motion - obtain $R\ and\ \Delta\theta$ like to the left
  - $\begin{pmatrix} x_{new} \\ y_{new} \\ \theta_{new} \end{pmatrix} = \begin{pmatrix} x + R\sin(\theta + \Delta\theta) - \sin\theta \\ y - R\cos(\theta + \Delta\theta) - \cos\theta \\ \theta + \Delta\theta \end{pmatrix}$

Uncertainty in Motion (2D):

- e, f, g are zero mean gaussian terms
  - Model the deviation from ideal path
- Straight line of distance D
  - $\begin{pmatrix} x_{new} \\ y_{new} \\ \theta_{new} \end{pmatrix} = \begin{pmatrix} x + (D + e)\cos\theta \\ y + (D + e)\sin\theta \\ \theta + f \end{pmatrix}$
- Pure rotation of angle α
  - $\begin{pmatrix} x_{new} \\ y_{new} \\ \theta_{new} \end{pmatrix} = \begin{pmatrix} x \\ y \\ \theta + \alpha + g \end{pmatrix}$

Gearing:

- Gear 1 has radius r_1 and torque t_1
  - exerts force $F = \frac{t_1}{r_1}$
  - has velocity $v = r_1\omega_1$
- Gear 2 has radius r_2 and torque t_2
  - $\boldsymbol{t_2 = r_2 F = \frac{r_2}{r_1} t_1}$
    - **torque ratio is radius_2 / radius_1**
  - $\boldsymbol{\omega_2 = \frac{v}{r_2} = \frac{r_1}{r_2}\omega_1}$
    - **angular v ratio is radius_1 / radius_2**

Motor Control:

- P - reduces error
- I - reduces steady-state error
- D - can reduce settling time
- Ziegler-Nichols method
  - find a k_u that shows oscillation (I=D=0)
  - k_p = 0.6k_u
  - k_i = 2k_p/Period_u
  - k_d = k_p*Period_u / 8

Position Based Path Planning:

- $\begin{pmatrix} d_x \\ d_y \end{pmatrix} = \begin{pmatrix} W_x - x \\ W_y - y \end{pmatrix}$
- Angle between current location and waypoint
  - $\alpha = \tan^{-1}\left(\frac{d_y}{d_x}\right)$
    - Python atan gives range $-\pi \le \alpha \le \pi$
- Angle to rotate:
  - $\beta = \alpha - \theta$
    - Range check this for more efficient rot
- After rotating, distance to move forward:
  - $D = \sqrt{d_x^2 + d_y^2}$

Sensors

23 October 2018 18:07

Types of Sensors:

- **Proprioceptive**
  - Self-sensing
  - Internal
  - Value of measurement depend only on current and previous internal robot states
- **Exteroceptive**
  - Outward-looking
  - Become *aware* of its environment
  - Value of measurement depends on state of robot **and** world around it

Touch Sensors:

- Binary on/off
- When switch closed, current flows, hit - 1
- **Bump** Detection:
  - Last line of defense
  - Needs immediate reaction
    - Evasive maneuver
    - Or just stop current motion
  - Can use a ring of touch sensors, determine direction of collision

Sonar Sensors:

- Emit an ultrasonic pulse and time until echo returns
- Max range - a few meters
- Can also use a ring of these

Laser Range Sensors:

- Return an array of depth measurements
  - From a scanning beam
- Sub-mm accuracy
- Normally bulky and expensive

Servoing:

- Robot controls coupled to a sensor reading
  - Updated regularly in a negative feedback loop
    - (closed control loop)
  - Updating frequency needs to be sufficiently high to avoid oscillating Behaviour from feedback loop
  - $v = -K_p(v_{desired} - v_{actual})$
    - Get v_{actual} from sensors
      - This is the direct coupling
    - Sensors can sometimes produce 'garbage' results
      - Can remove this - e.g. median filtering
      - Can reduce system responsive if using filtering

Some Examples:

- Follow an object to collide
  - $s = K_p\alpha$
- Avoid a collision with an object
  - $s = K_p(\alpha - \sin^{-1}\frac{R}{D})$

Probabilistic Sensor Modelling:

- Sensor does not report exact truth
- Need to characterize uncertainty of the sensor
  - Then can build a probabilistic model for it to use
  - $p(z_0|x, y)$
    - Probability of measurement being correct given current state of robot and world

Likelihood Functions:

- Probability of measuring Z given that I'm expecting a ground truth value of M
- Narrow Gaussian band around M
  - With a SMALL constant K added to all probabilities
    - Represents a fixed percentage of garbage results
    - More robust

Probabilistic Robotics

23 November 2018 11:37

Classical AI falls down in real world as data obtained from sensors is inaccurate.
-> need to acknowledge uncertainty and model to extract useful info

Sensor Fusion: combining data from many different sources into useful information

Bayesian Probabilistic Inference:

- Measure of subjective belief
- **Bayes' Rule**
  - $P(X|Z) = \frac{P(Z|X)P(X)}{P(Z)}$
  - $\boldsymbol{posterior} = \frac{\boldsymbol{likelihood} * \boldsymbol{prior}}{marginal\ likelihood}$
- Represent prior and likelihood as gaussian,
  - When multiplied they produce a tighter band
    - i.e. posterior narrowest

Particle Representation:

- Each particle has attributes + weight
- $\Sigma weights = 1$
  - normalized
- Pros
  - Simple
  - Represent and shape distribution
    - Even multi-peak dist.
- If too many particles
  - Computationally expensive
- Too little
  - Poor ability to represent shape of distribution

Particle Filtering:

1. **Motion Prediction** based on proprioceptive sensors
   a. When robot moves
   b. Pass particles through a function which update their position based on the deterministic input and a random component
2. **Measurement Update** based on exteroceptive sensors
   a. Applying Bayes' Rule to each particle
   b. $w_{i_{new}} = P(z|x_i) * w_i$
      i. Posterior = likelihood * prior
      ii. Don't need to worry about P(z) as it will be removed in (3)
3. Normalisation
4. Resampling

Monte Carlo Localisation

23 November 2018 11:38

MCL
- A Bayesian probabilistic filter
- 'Survival of the fittest'

Continuous Localisation:
- Tracking
- Given good estimate of last position
- Estimate new position after new measurements
- All particles set to start position with equal weight

Global Localisation:
- 'kidnapped robot problem'
- Environment is known
- But position is completely uncertain
- Find the position
- Particle state set randomly, all have equal weight

Inferring an Estimate:
- $\bar{\boldsymbol{x}} = \sum_{i=1}^{N} w_i \boldsymbol{x}_i$

MCL/Particle Filter:
1. Motion Prediction based on proprioceptive sensors
   a. Watch out for angular-wrap around
2. **Measurement Update** based on exteroceptive sensors
   a. Likelihood function - sonar
   b. Also can use a Compass Sensor
3. **Normalisation**
   a. $sum\ of\ all\ weights\ should\ add\ to\ 1$
   b. $w_{i(new)} = \frac{w_i}{\sum_{i=1}^{N} w_i}$
4. **Resampling**
   a. Generate a new set of N particles with equal weights
   b. But spatial distribution now reflects probability density
   c. **Cumulative probability distribution**
      i. Generate a random number R= 0 to 1
      ii. Then duplicate particle whose cumulative probability range covers R
   d. Can skip (3) and resample directly

Likelihood function - **Sonar:**
- Have particles current state **x**
- Have wall coordinates $\left(A_x, A_y\right)$ and $\left(B_x, B_y\right)$
- Distance between particle and 'infinite wall'
  - $\boldsymbol{m} = \frac{\left(\boldsymbol{B_y} - \boldsymbol{A_y}\right)\left(\boldsymbol{A_x} - \boldsymbol{x}\right) - \left(\boldsymbol{B_x} - \boldsymbol{A_x}\right)\left(\boldsymbol{A_y} - \boldsymbol{y}\right)}{\left(\boldsymbol{B_y} - \boldsymbol{A_y}\right)\boldsymbol{cos\theta} - \left(\boldsymbol{B_x} - \boldsymbol{A_x}\right)\boldsymbol{sin\theta}}$
- Have sonar measurement z
  - Range check m
    - $\nless 0$
    - $\begin{pmatrix} x + mcos\theta \\ y + msin\theta \end{pmatrix}$ $lies\ within\ wall\ boundaries$
- $p(z|m) \propto \boldsymbol{e}^{\frac{-(\boldsymbol{z}-\boldsymbol{m})^2}{2\sigma_s^2}}$
  - Get prob of z given m
  - Modelled by gaussian distribution
    - $\sigma_s\ - sonar\ s.d.$
- Add a very small constant K
  - Constant probability of garbage value, uniformly distributed across range of the sensor
  - Less aggressive in killing off particles which are unlikely
  - Occasional garbage result won't lead to good particles dying off

- If angle between sonar and normal to wall is too great
  - Might want to ignore as too large an angle can give inaccurate reading
  - $\beta = \cos^{-1}\left(\frac{cos\theta\left(A_y - B_y\right) + sin\theta\left(B_x - A_x\right)}{\sqrt{\left(A_y - B_y\right)^2 + \left(B_x - A_x\right)^2}}\right)$

(2) with a **Compass Sensor:**
- Digital Compass - estimate rotation without drift
- $measure\ bearing\ \beta\ relative\ to\ north$
- $P\left(\beta\middle|\boldsymbol{x}_i\right)$?
- Define γ as bearing of x-axis of frame
- $\boldsymbol{\beta = \gamma - \theta}$ $if\ \beta working\ perfectly$
- So
  - $P\left(\beta\middle|\boldsymbol{x}_i\right) \propto \boldsymbol{e}^{\frac{-(\boldsymbol{\beta}-(\boldsymbol{\gamma}-\boldsymbol{\theta}))^2}{2\sigma_c^2}}$
- Particles far with theta far from right orientation get low weights

Advanced Sonar Sensing

23 November 2018 11:38

Global Localisation via Recognition:

- **Learn** location beforehand
  - Take many measurements
  - Characterize this location
    - Form a 'signature'
- Robot can only recognise previously learnt locations

Measuring a Location:

- Place robot at target
- Take measurements
- Raw values - signature

Place Recognition:

- Take measurements at new position
- Compare against all signatures
  - New sig H_m
  - Saved sig H_i
  - $D_i = \sum_j (H_m[j] - H_i[j])^2$
- Choose lowest D_i
  - But $D_i > t$
    - $t = threshold$
    - So not in unknown location
- Match this new histogram for all locations
- Once found best matching depth signature
  - Shift measured to find orientation at this one location

Probabilistic Occupancy Grid Mapping:

- Building a 'map' when we know location
- Occupancy Grid Map Representation:
  - Square grid of area we'd like to map
  - Each cell represents probability that it's occupied by an obstacle
    - $P(O_i)$
      - Initialise to 0.5
      - 1 - black - occupied
      - 0 - white - empty
  - $P(E_i) = 1 - P(O_i)$
- **Update after sonar measurement**
  - Get reading d
  - Cells further than d likely to be occupied
  - Cells close than d, and in the direction of d, likely to be empty
    - Width of around 10-15 deg free
- **Find log odds update U:**
  - If cell within d and width of sonar
  - Add a *constant negative value*
  - If around d, and within sonar
  - Add a *constant positive value*
  - Assuming cells occupancies are independent of other cells
- Thresholds to determine if empty or occupied
- Memory intensive
- Subject to drift due to localisation uncertainty

- **Bayesian Update of Occupancy**
  - $P(O_i|Z) = \frac{P(Z|O_i)P(O_i)}{P(Z)}$
  - $P(E_i|Z) = \frac{P(Z|E_i)P(E_i)}{P(Z)}$
  - $P(O_i|Z) + P(E_i|Z) = 1$
    - So can normalize
  - $\frac{P(O_i|Z)}{P(E_i|Z)} = \frac{P(Z|O_i)P(O_i)}{P(Z)} \frac{P(Z)}{P(Z|E_i)P(E_i)}$
  - $\frac{P(O_i|Z)}{P(E_i|Z)} = \frac{P(Z|O_i)P(O_i)}{P(Z|E_i)P(E_i)}$
  -
  - Odd notation: $o(A) = \frac{P(A)}{P(\bar{A})}$
  - $o(O_i|Z) = \frac{P(Z|O_i)}{P(Z|E_i)} * o(O_i)$
  - $\ln o(O_i|Z) = \ln\left(\frac{P(Z|O_i)}{P(Z|E_i)}\right) + \ln o(O_i)$

    - Can now update by just adding
    - Prob 0.5 = 0 log odds

Logs odds	Prob
>0	>0.5
<0	<0.5

- Usually cap log odds +-

SLAM

23 November 2018 11:38

<u>**When** to use SLAM:</u>

- autonomous
- no prior map
- no artificial beacons or GPS
- need to determine robot's location

Incrementally build a map.

consisting of natural scene ***features***

Data Association - matching features

distinctive

recognisable from different viewpoints

Localise with respect to the map.

<u>Main assumption:</u>

- world is **static**
- probabilistic estimation to the features
- **joint distribution** of robot and mapped world
  - features are added, so dimension of this distribution will grow
  - represent as a joint Gaussian
  - Updates made via **Extended Kalman Filter**
- **State Vector** and **Covariance Matrix**
  - SV - robot state is first element
  - SV - additional element for features
  - CM - square matrix - covariance between all elements

<u>SLAM Process:</u>

a. Measure a point A
   i. this has some uncertainty
   ii. Robot relative position - certain
b. Robot moves to X
   i. robot position on the grid in uncertain
c. Robot measures points B and C
   i. they **inherit** robots uncertainty
   ii. plus measurement uncertainty
d. Robot moves back to starting point
   i. robot position uncertainty grows more
e. Robot re-measures A
   i. **mini loop closure**
   ii. Uncertainty for all shrink
f. Robot re-measures B
   i. Uncertainty for all shrink

<u>Limitations of Metric SLAM:</u>

Limits to small domains due to:

- computational intensive for large joint PDFs
- bad at large distances
  - growth in uncertainty
- data association hard with large uncertainty

<u>Large Scale SLAM:</u>

- *metric/topologica*l approximation of full metric
- Place Recognition - perform **loop closure**
- **Map Relaxation** - optimise a map after loop closure
- Loop closure detection:
  - save invariant signatures at regular intervals
  - compare new measurements vs these saved signatures
  - if loop found
    - add constraint to graph
- Relaxation
  - **pose graph optimisation**
  - set of node positions which are maximally probable s.t.
    - metric constraints
    - topological constraints

Planning

05 December 2018 14:21

- Map of environment known
- Plan a path around a set of obstacles to reach a target

Local Planning - Dynamic Window Approach:

- for each possible motion the robot can make in time **dt**
- calculate **cost/benefit** on distance from target and obstacles
  - if making that movement for **longer time $\boldsymbol{\tau}$**
- choose best and **execute for dt**, then repeat

DWA for Differential Drive:

- can adjust $\boldsymbol{v_L\ and\ v_R}$ up to a max value
- **9 possibilities**, either can go up, down or stay same
- Use formulas to calc position of robot for all possibilities

Benefit:

- $D_F = \sqrt{(T_x - x)^2 + (T_y - y)^2} - \sqrt{(T_x - x_{new})^2 + (T_y - y_{new})^2}$

- Benefit: $\boldsymbol{B = W_B * D_F}$

Cost:

- **subtracted** from benefit
- only **closest obstacle** considered, found by searching
- $C = W_C * \left(D_{safe} - \left(\sqrt{(O_x - x_{new})^2 + (O_y - y_{new})^2} - r_{robot} - r_{obstacle}\right)\right)$
  - $D_{safe} - distance\ to\ stay\ away$

Decision:

- choose path with $\mathbf{max}\,\boldsymbol{B - C}$
- execute motion for **dt**

Accurate Motion

25 November 2018 19:55

<u>Tuning:</u>

- PID - same for both
- Feed-forward and minPWM separately
  - Feed-forward - compensate for weaker motor
    - i.e. if drifting left heavily
      - Right stronger than left
        - Increase left motor FF

<u>Moving in straight line:</u>

- <u>Think of wheel motion</u>
- Convert distance into revolutions of wheel
- $revs = \frac{dist}{2\pi r_w}$
- $angle\ to\ increase: 2\pi * revs$
  - $\boldsymbol{angle = \frac{dist}{r_w}}$

To move forward:

- Increase angle for both wheels
  - $increaseMototAngeReferences(motors, [angle, angle])$

To move backward:

- Decrease angle for both wheels
  - $increaseMototAngeReferences(motors, [-angle, -angle])$

<u>Covariance Matrix:</u>

- $\begin{bmatrix} \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2 & \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2(y_i - \bar{y})^2 \\ \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2(y_i - \bar{y})^2 & \frac{1}{N}\sum_{i=1}^{N}(y_i - \bar{y})^2 \end{bmatrix}$

<u>Turn on the spot by angle β:</u>

- Think of robot spinning like a wheel
- Radius of robot = half of wheel base = R
- Find distance moved
  - $D = revs\ of\ robot\ * circumference\ of\ turning\ circle$
  - $D = \frac{\beta}{2\pi} * 2\pi R = \beta R$
- Translate this into revolutions for the wheel
- $revs_w = \frac{dist}{2\pi r_w} = \frac{\beta R}{2\pi r_w}$
- Convert this into angle to change wheel by
- $\alpha = 2\pi * revs_w = 2\pi * \frac{\beta R}{2\pi r_w}$
- $\alpha = \frac{\beta R}{r_w} = \boldsymbol{angle\ to\ turn\ robot * \frac{robot\ radius}{wheel\ radius}}$

Increase for direction we want to go, decrease the opposite

To turn right:

- $increaseMototAngeReferences(motors, [-angle, angle])$

To turn left:

- $increaseMototAngeReferences(motors, [angle, -\ angle])$

Investigating Sensors

30 November 2018 15:17

Touch Sensors:

In while loop

1. **Read** touch sensor values
2. **First** if any hit
   - Stop movement
   - Else
     - Scan again (top loop)
3. **If** both hit
   - Turn back, left or right
   - Else if left hit
     - Reverse, turn right
   - Else if right hit
     - Reverse, turn left
4. **Continue** moving forward
   - (maintain speed)

Use time.sleep(x) between different movements

Sonar Sensor:

1. Maintain forward distance
   - 30 cm distance to wall in front
   - Based on sonar reading, either move forward or backward
   - If further away from desired value, move quicker
   - As you get closer, move slower
   - $\boldsymbol{velocity} = \boldsymbol{-K_p(30 - sonar_{dist})}$
     - Choose K_p for smooth motion - experiment

2. Follow Wall:
   - 30cm distance to wall on *right* (or left)
   - If far away, left should speed up and right slow down
     - Turn right, towards wall
   - If too close, left should slow down and right speed up
     - Turn left, away from wall
   - $base_v$: $avg\ speed\ to\ maintain\ when\ 30cm\ from\ wall$
   - $error = -K_p(30 - sonar_{dist})$
   - $left_v = base_v - error$
   - $right_v = base_v + error$
     - Change sign for maintain to the *left*

Probabilistic Motion and Sensing

02 December 2018 13:37

Implementing Motion Prediction:

- Create pre-allocated arrays in Python
  - Size = NUMBER_OF_PARTICLES
- Choose suitable e, f, g values
  - For 2D motion - straight angle and pure rotation
  - e and f should give a 'banana' shape at end of line
    - Usually e>f
  - In Python
    - $random.guass(mu, \sigma)$
    - Picks a random number

Sonar Investigation:

- Around 10% garbage results
  - So median filter over 10
- After about 45 degrees, Sonar fails to give accurate reading
- No real systematic error
- Only useable form around 20cm
  - Very bad lower than that
    - But systematic error
      - Not random
  - Max range ~2m

Monte Carlo Localisation

02 December 2018 14:55

Implementing calculate_likelihood:

- For all particles
  - $w_i *= calc\_likelihood(x, y, \theta)$

1. Account for z
   a. Placement away from center
   b. Systematic errors
2. $c = \cos(\theta)$
   $s = \sin(\theta)$
   a. Check for either $c\ or\ s == 0$
   b. Calculate '**m**' for all walls
3. **Find best m**
   a. Greater than 0
   b. Smaller than rest
   c. Fits inside wall boundaries
      i. sort boundaries first
      ii. (check for small > larger)
4. Case:
   a. No feasible m - return 0
5. **Read of gaussian function**
   a. Mean = m
   b. Sd = s.d. of sonar
6. Add K
   a. Return value

Implementing Resampling:

1. Cumulative weight array
   a. $cw[0] = w[0]$
   b. $for\ i\ in\ range(1, N)$:
      $\boldsymbol{cw[i] = cw[i-1] + w[i]}$
2. New **temp arrays**
   a. For each state variable
   b. Each of size N
3. Select N new particles
   a. Get random number
   b. $for\ j\ in\ range(N)$:
      $if\ \boldsymbol{cw[j] - rand \geq 0}$
      $clone\ this\ particle$
      $break$
4. Overwrite old arrays with temp arrays
5. Give all particles **equal weights**
   $w[i] = 1/N$

Navigate to Waypoint:

While not at Waypoint (X,Y):

- $X_{diff} = X - estimate_x$
- $Y_{diff} = Y - esttimate_y$
- $\boldsymbol{dist} = \sqrt{X_{diff}^2 + Y_{diff}^2}$
- $angle_{dest} = atan2\left(Y_{diff}, X_{diff}\right)$
  - Returns angle $-\pi \leq \theta \leq \pi$
- $\boldsymbol{angle_{rot}} = angle_{dest} - estimate_\theta$

1. Rotate
   a. Update particles angles
   b. Add $angle_{rot} + error$
   c. Update $estimate_\theta$
2. Move forward
   a. Update particles
3. **Measurement Update**
   a. Read Sonar
   b. Calculate likelihood and update weights for all particles
   c. Normalise
   d. Resample
4. Update estimate of robot position

Place Recognition

02 December 2018 14:55

<u>Circular Scan:</u>

for loop

- take reading
- spin $\phi\ to\ left$
  - $\phi = \frac{2\pi}{n}$
  - use $left_{90}\left(\frac{\phi}{\frac{\pi}{2}}\right)$

<u>Compare Signatures:</u>

- sum of differences squares

<u>Make Depth Histograms:</u>

- create array of n bins
- v=**floor**(depth/bin size)
- histogram[v]+=1

<u>Learn Location:</u>

- perform circular scan
- storing measurements into array(n)
- allocate unique idx to signature

<u>Recognise Location:</u>

1. Circular scan
2. make depth histograms for measurement and known
   - compare these
   - find best match
3. find rotation ->

<u>Angular Shift:</u>

- shifted = []
  - init to angle vs depth
- for len(shifted)
  - compare shifted and predicted
  - if dist < best_dist
    - update best_dist
    - $angle = i * degrees\ per\ shift$
  - right shift shifted
- if $angle > \pi$
  - $angle\ -= 2\pi$
- return angle