Kinodynamic Motion Planning

Lecture 4

Kinodynamic Motion Planning

- Planning in robotics is often not just a geometrically feasible path from one location to another. When kinematic or dynamic constraints are present, planning becomes non-trivial. The simple car-parking problem illustrates this.

Introduction (contd.)

Kinematics

kinematics	dynamics
The effect of a robot's geometry on its motion.	The effect of all forces (internal and external) on a robot's motion.
If the motors move ***this*** much, where will the robot be?	If the motors apply ***this*** much force, where will the robot be?
Assumes that we control ***encoder readings***…	Assumes that we control ***motor current***…

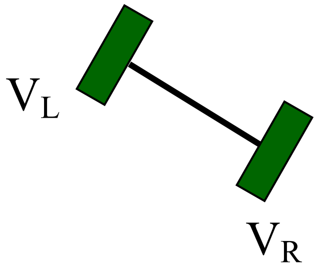
Differential drive

Most common kinematic choice

Most miniature robots…

ER1, Pioneer, Rug warrior

- difference in wheels' speeds determines its turning angle



Questions (forward kinematics)

Given the wheel's velocities or positions, what is the robot's velocity/position ?

Are there any inherent system constraints?

1) Specify system measurements

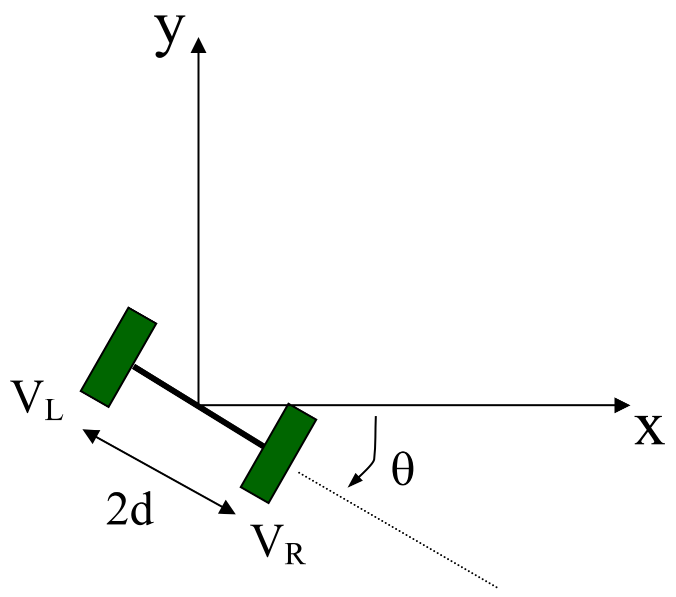
2) Determine the point (the radius) around which the robot is turning.

3) Determine the speed at which the robot is turning to obtain the robot velocity.

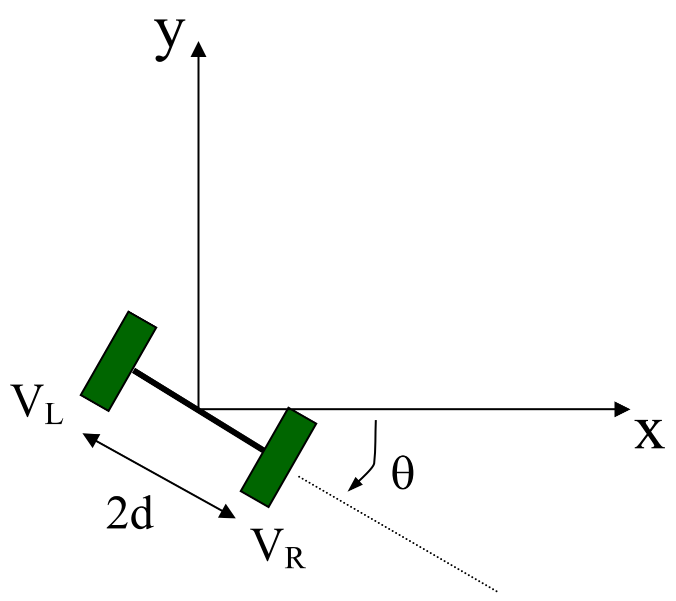
4) Integrate to find position.

Differential drive

1) Specify system measurements
   - consider possible coordinate systems



Differential drive

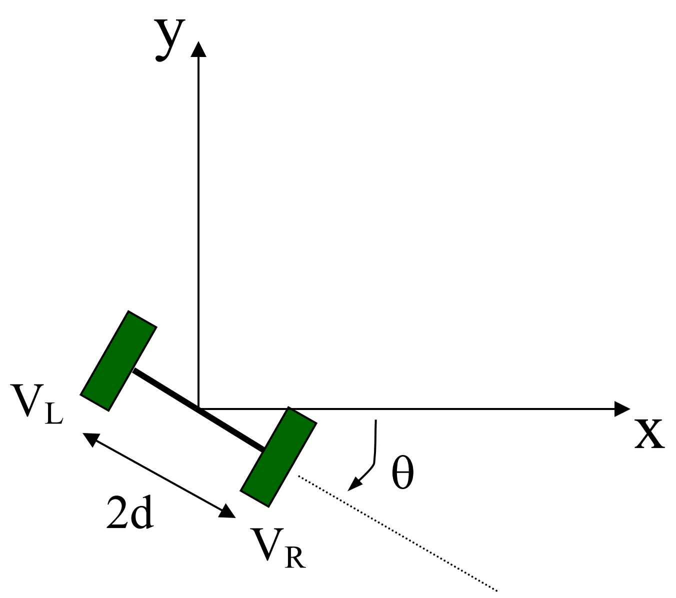


1) Specify system measurements
- consider possible coordinate systems

---

2) Determine the point (the radius) around which the robot is turning.

Differential drive



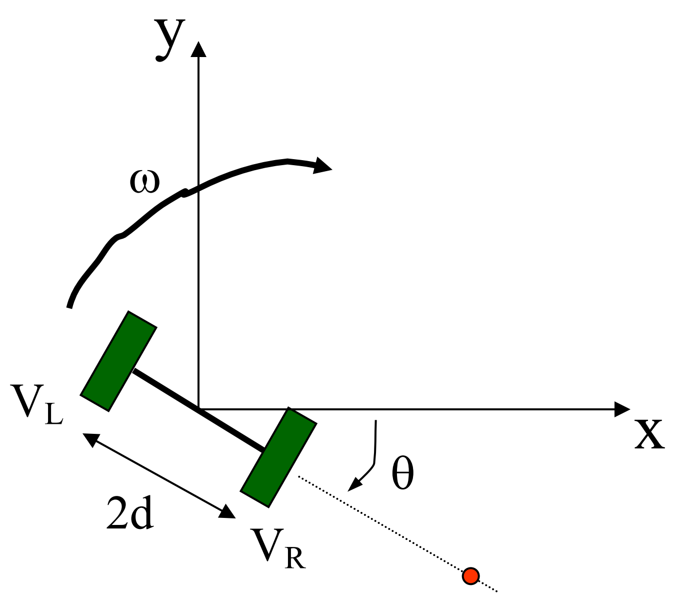
1) Specify system measurements
   - consider possible coordinate systems

2) Determine the point (the radius) around which the robot is turning.

- to minimize wheel slippage or lateral motion, the instantaneous center of curvature (the **ICC**) must lie at the intersection of the wheels' axles

- each wheel must be traveling at the same angular velocity

Differential drive



ICC "instantaneous center of curvature"

1) Specify system measurements
   - consider possible coordinate systems

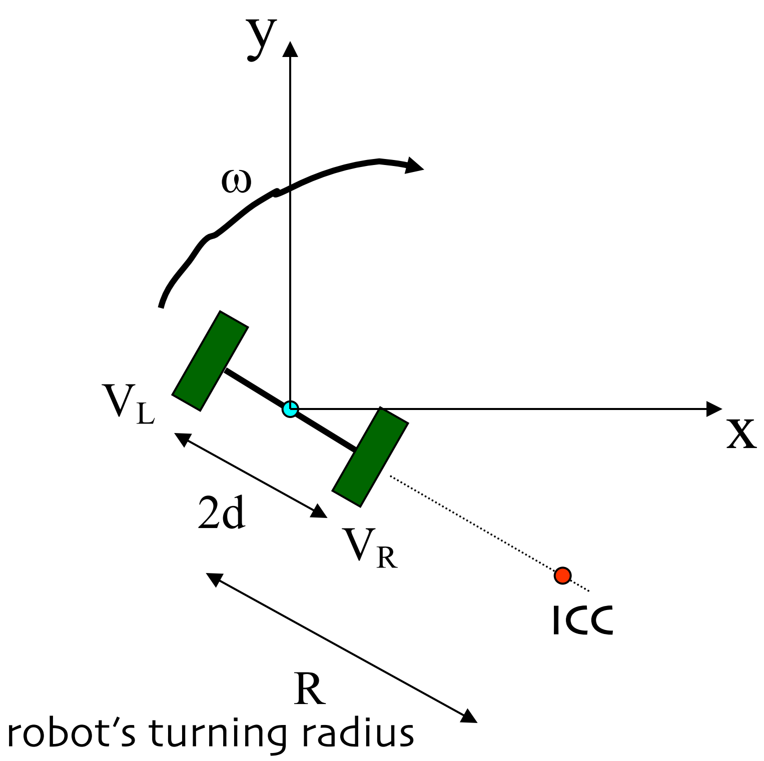
2) Determine the point (the radius) around which the robot is turning.

- to minimize wheel slippage, this point (the **ICC**) must lie at the intersection of the wheels' axles

- each wheel must be traveling at the same angular velocity **around the ICC**

(assume the wheel diameter is accounted for already)

Differential drive



1) Specify system measurements
   - consider possible coordinate systems

2) Determine the point (the radius) around which the robot is turning.

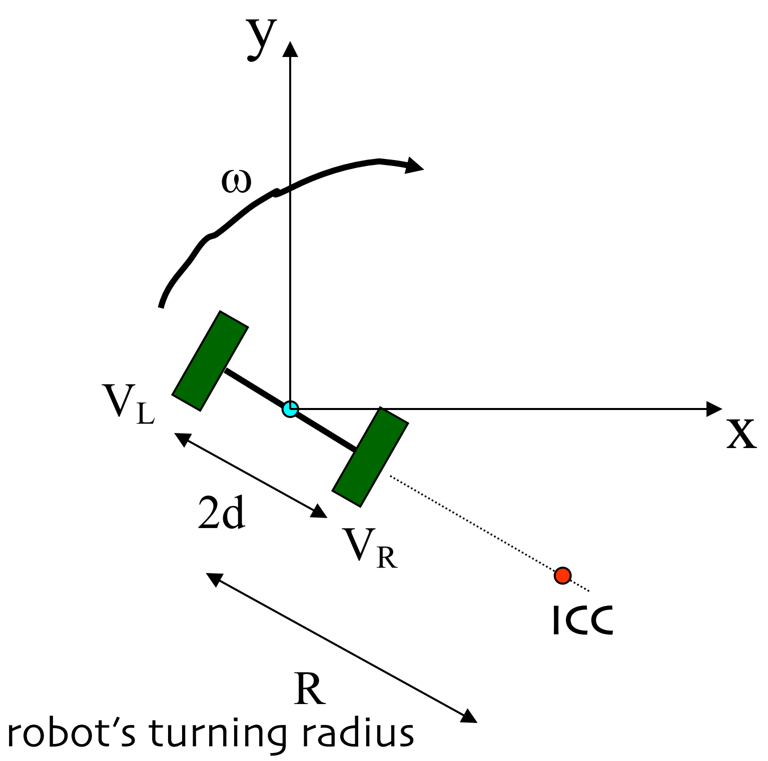
   - each wheel must be traveling at the same angular velocity around the ICC

3) Determine the robot's speed around the ICC and its linear velocity

$$\omega(R+d) = V_L$$

$$\omega(R-d) = V_R$$

Differential drive



1) Specify system measurements
- consider possible coordinate systems

2) Determine the point (the radius) around which the robot is turning.

- each wheel must be traveling at the same angular velocity around the ICC

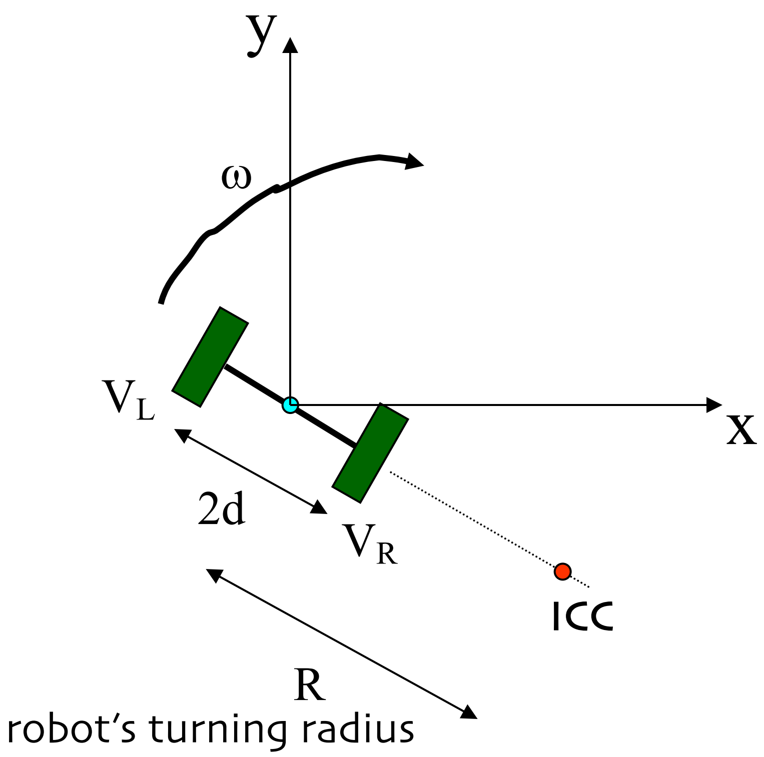
3) Determine the robot's speed around the ICC and its linear velocity

$$\omega(R+d) = V_L$$

$$\omega(R-d) = V_R$$

of these five, what's known & what's not?

Differential drive



1) Specify system measurements
   - consider possible coordinate systems

2) Determine the point (the radius) around which the robot is turning.

   - each wheel must be traveling at the same angular velocity around the ICC

3) Determine the robot's speed around the ICC and its linear velocity

$$\omega(R+d) = V_L$$

$$\omega(R-d) = V_R$$

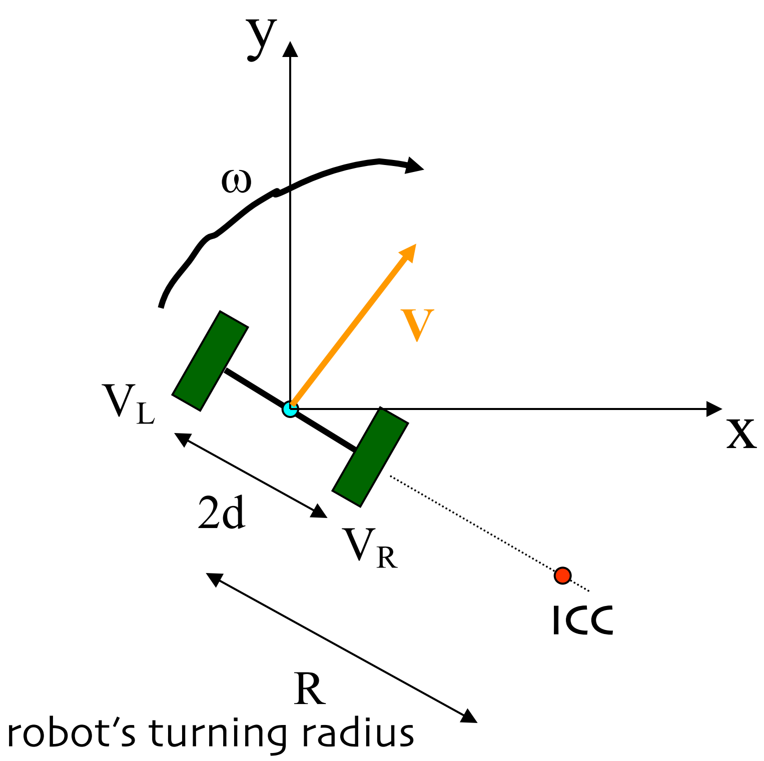
Thus,

$$\omega = (V_R - V_L) / 2d$$

$$R = d(V_R + V_L) / (V_R - V_L)$$

are there interesting cases?

Differential drive



1) Specify system measurements
   - consider possible coordinate systems

---

2) Determine the point (the radius) around which the robot is turning.

   - each wheel must be traveling at the same angular velocity **around the ICC**

---

3) Determine the robot's speed around the ICC and its linear velocity

$$\omega(R+d) = V_L$$

$$\omega(R-d) = V_R$$

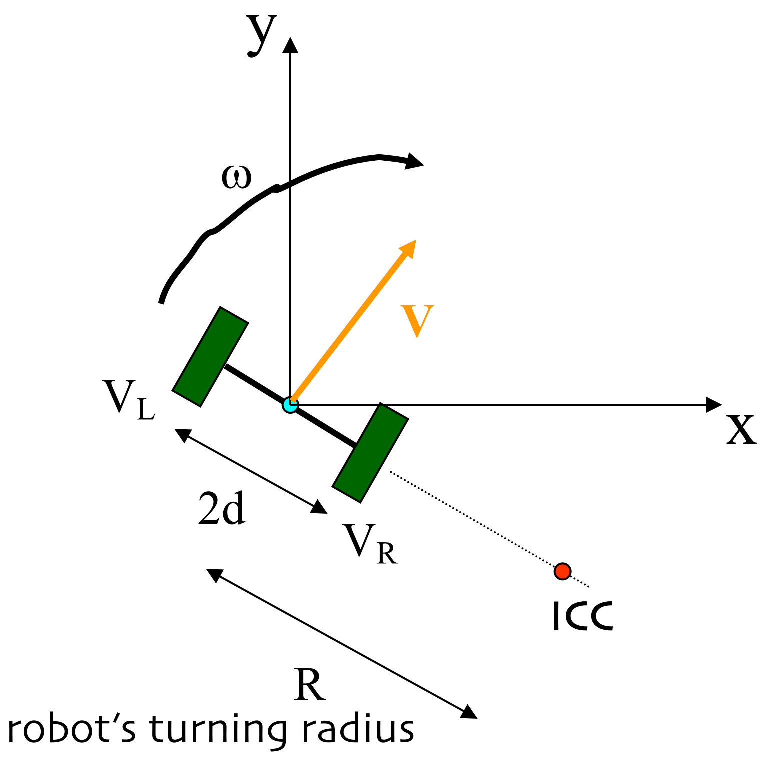
Thus,

$$\omega = (V_R - V_L) / 2d$$

$$R = d(V_R + V_L) / (V_R - V_L)$$

So, what is the robot's velocity?

Differential drive



1) Specify system measurements
- consider possible coordinate systems

2) Determine the point (the radius) around which the robot is turning.
- each wheel must be traveling at the same angular velocity around the ICC

3) Determine the robot's speed around the ICC and its linear velocity

$$\omega(R+d) = V_L$$

$$\omega(R-d) = V_R$$

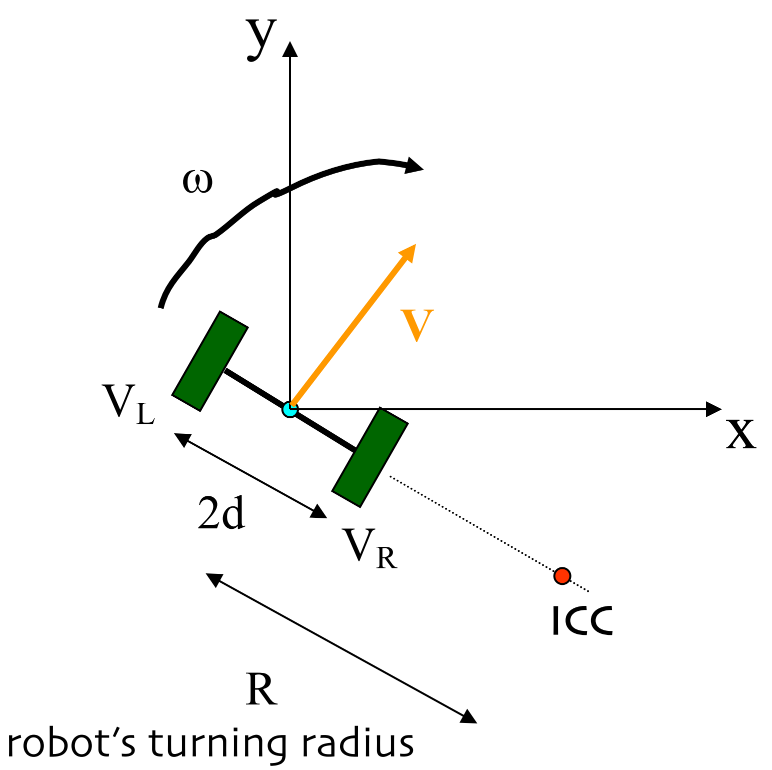
Thus,

$$\omega = (V_R - V_L) / 2d$$

$$R = d(V_R + V_L) / (V_R - V_L)$$

So, the robot's velocity is $V = \omega R = (V_R + V_L) / 2$

Differential drive



4) Integrate to obtain position

$$V_x = V \quad \cos(\theta \quad)$$

$$V_y = V \quad \sin(\theta \quad)$$

with

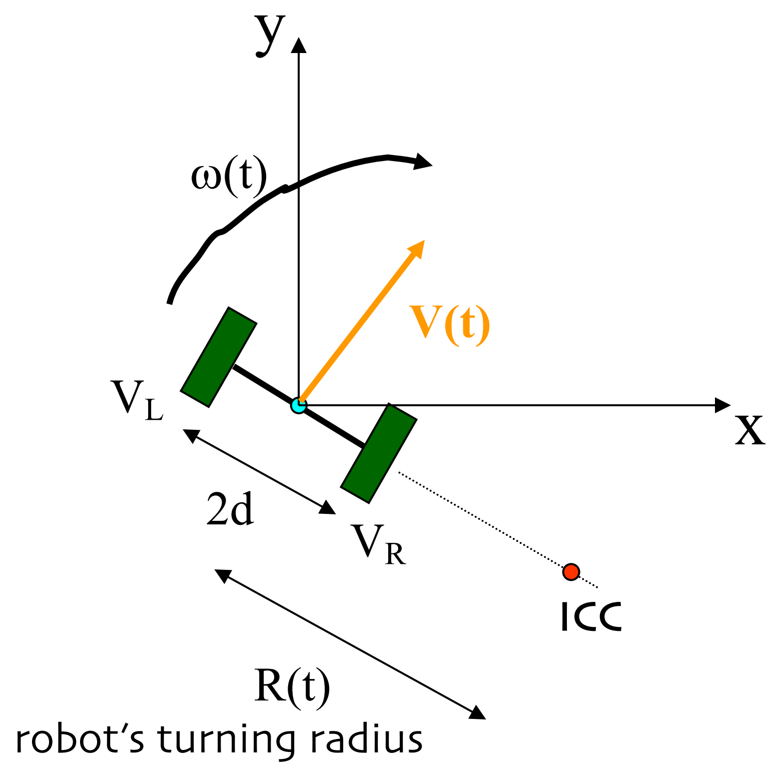
$$\omega = (V_R - V_L) / 2d$$

$$R = d (V_R + V_L) / (V_R - V_L)$$

$$V = \omega R = (V_R + V_L) / 2$$

What has to happen to change the ICC ?

Differential drive



4) Integrate to obtain position

$$V_x = V(t)\cos(\theta(t))$$

$$V_y = V(t)\sin(\theta(t))$$

with

$$\omega = (V_R - V_L) / 2d$$

$$R = d\,(V_R + V_L) / (V_R - V_L)$$

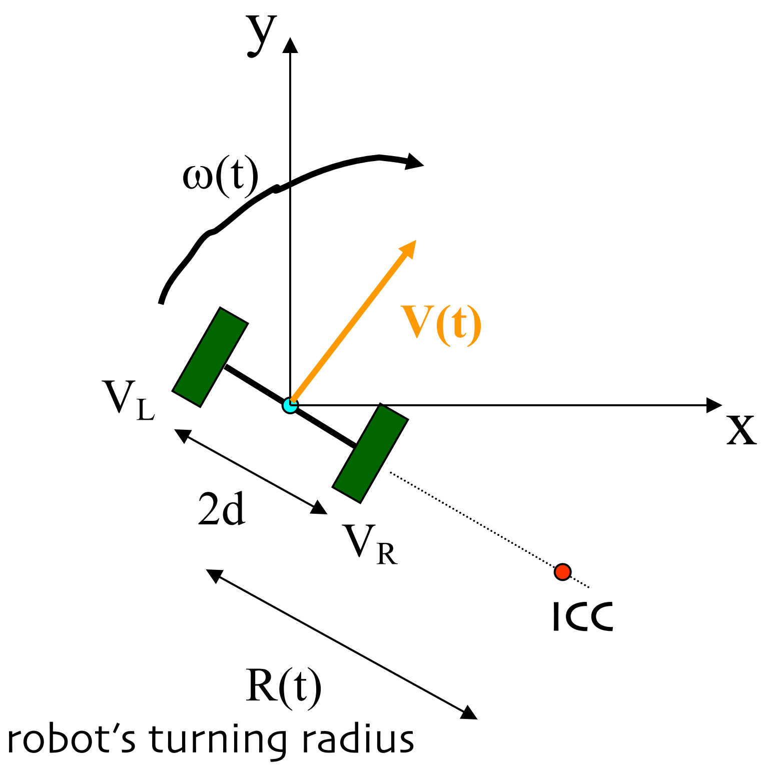
$$V = \omega R = (V_R + V_L) / 2$$

What has to happen to change the ICC ?

things have to change over time, **t**

Differential drive

4) Integrate to obtain position



$$V_x = V(t)\cos(\theta(t))$$

$$V_y = V(t)\sin(\theta(t))$$

Thus,

$$x(t) = \int V(t)\cos(\theta(t))\,dt$$

$$y(t) = \int V(t)\sin(\theta(t))\,dt$$

$$\theta(t) = \int \omega(t)\,dt$$

with

$$\omega = (V_R - V_L) / 2d$$

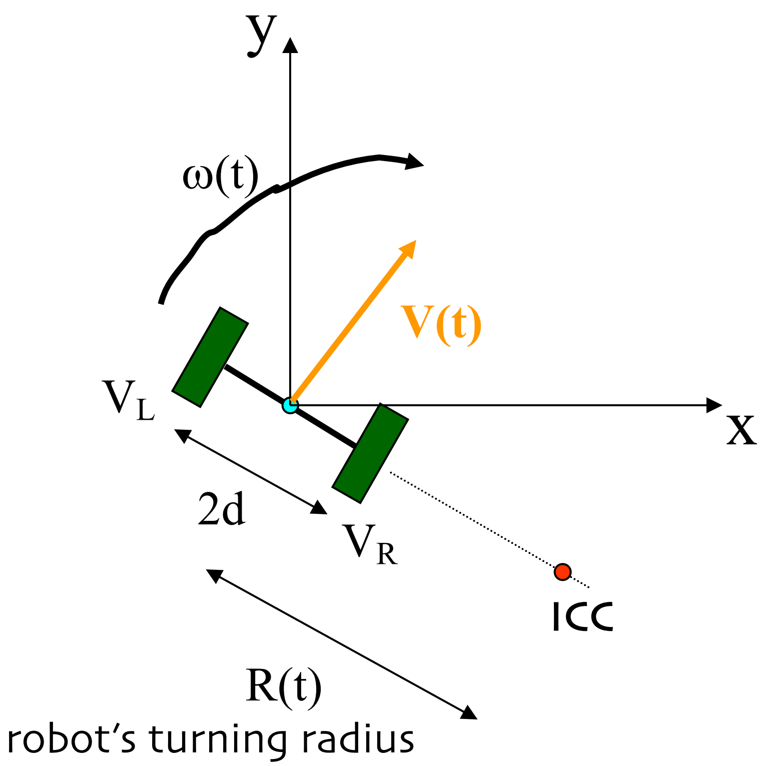
$$R = d\,(V_R + V_L) / (V_R - V_L)$$

$$V = \omega R = (V_R + V_L) / 2$$

What has to happen to change the ICC ?

things have to change over time, **t**

Differential drive



4) Integrate to obtain position

$$V_x = V(t)\cos(\theta(t))$$

$$V_y = V(t)\sin(\theta(t))$$

Thus,

$$x(t) = \int V(t)\cos(\theta(t))\,dt$$

$$y(t) = \int V(t)\sin(\theta(t))\,dt$$

$$\theta(t) = \int \omega(t)\,dt$$

Kinematics

with

$$\omega = (V_R - V_L) / 2d$$

$$R = d\,(V_R + V_L) / (V_R - V_L)$$

$$V = \omega R = (V_R + V_L) / 2$$

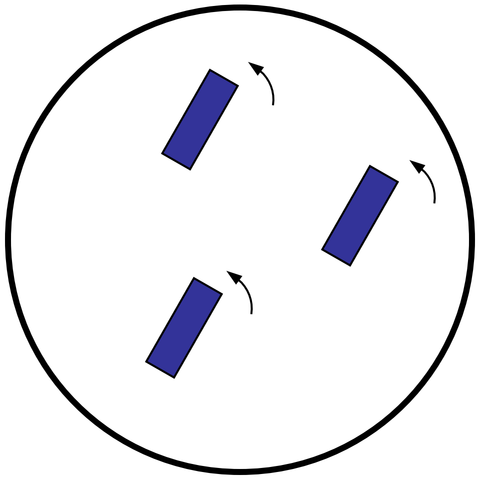
What has to happen to change the ICC ?

things have to change over time, **t**

Synchro drive

Nomad 200

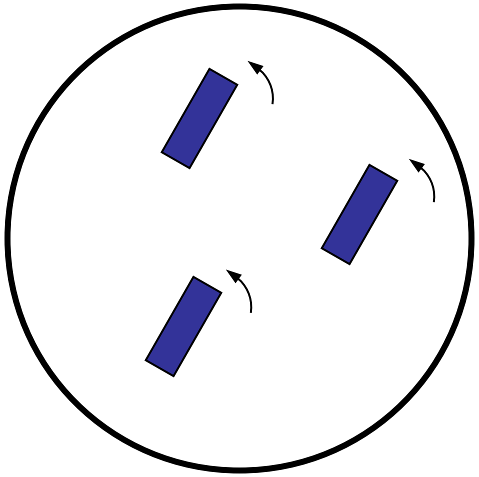
wheels rotate in tandem and remain parallel
all of the wheels are driven at the same speed



Synchro drive

Nomad 200

wheels rotate in tandem and remain parallel
all of the wheels are driven at the same speed



Where is the ICC ?

Questions (forward kinematics)

Given the wheel's velocities or positions, what is the robot's velocity/position ?

Are there any inherent system constraints?

1) Specify system measurements

2) Determine the point (the radius) around which the robot is turning.

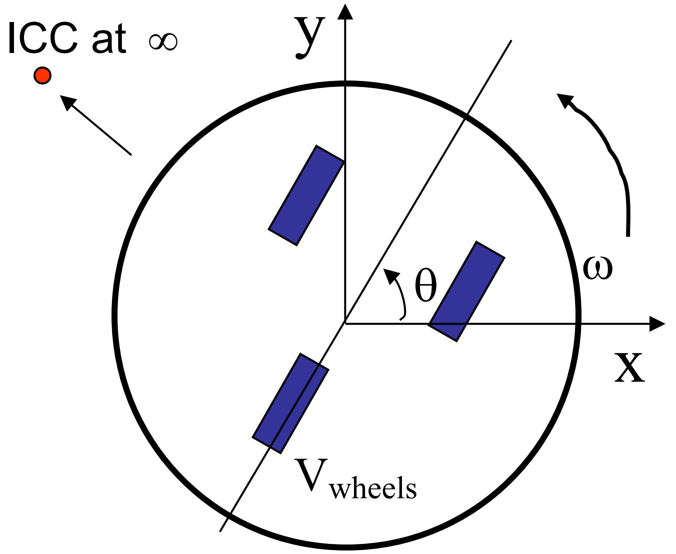
3) Determine the speed at which the robot is turning to obtain the robot velocity.

4) Integrate to find position.

Synchro drive

Nomad 200

wheels rotate in tandem and remain parallel
all of the wheels are driven at the same speed



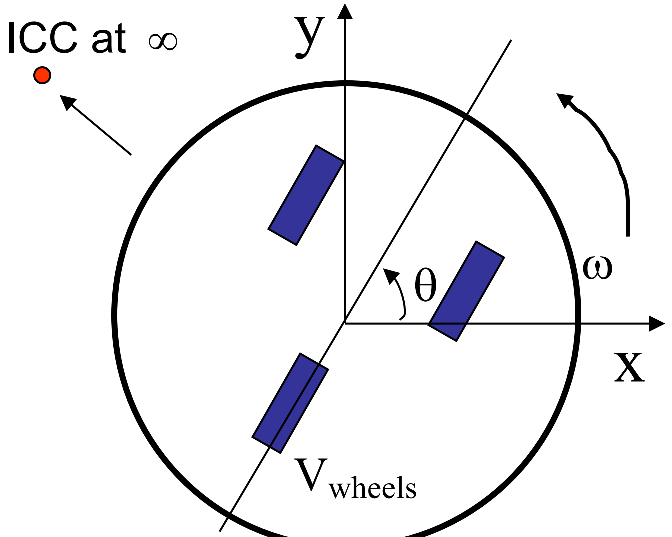
$$\left.\begin{aligned} V_{robot} &= V_{wheels} \\ \omega_{robot} &= \omega_{wheels} \end{aligned}\right\} \text{velocity}$$

$$\left.\begin{aligned} \theta(t) &= \int \omega(t)\, dt \\ x(t) &= \int V_{wheels}(t) \cos(\theta(t))\, dt \\ y(t) &= \int V_{wheels}(t) \sin(\theta(t))\, dt \end{aligned}\right\} \text{position}$$

Synchro drive

Nomad 200

wheels rotate in tandem and remain parallel
all of the wheels are driven at the same speed



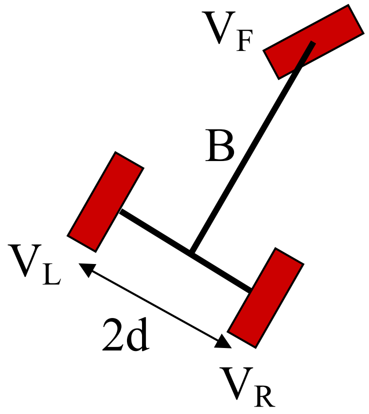
$$\left.\begin{aligned} V_{robot} &= V_{wheels} \\ \omega_{robot} &= \omega_{wheels} \end{aligned}\right\} \text{velocity}$$

Kinematics

$$\left.\begin{aligned} \theta(t) &= \int \omega(t)\, dt \\ x(t) &= \int V_{wheels}(t) \cos(\theta(t))\, dt \\ y(t) &= \int V_{wheels}(t) \sin(\theta(t))\, dt \end{aligned}\right\} \text{position}$$

Tricycle drive

Mecos tricycle-drive robot



- The velocity of the front wheel is V_F

  (i.e., the conversion from angular velocity with wheel diameter is already done…)

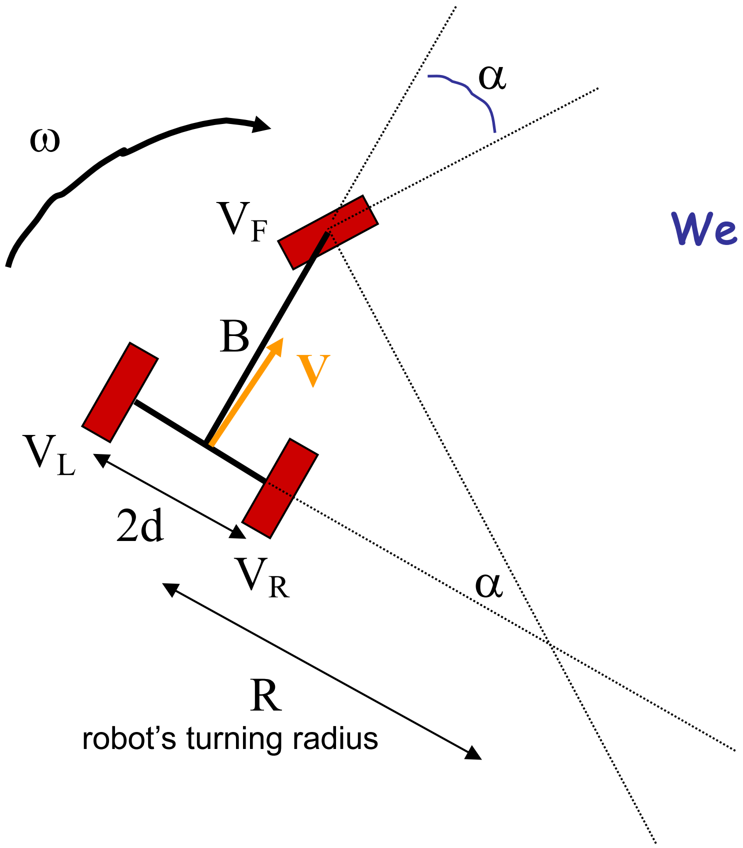
- We know the distances 2d and B

- What else do we need to know?
- Where is the ICC?
- How fast is the trikebot rotating around it?
- What are V_L and V_R ?

Mecos tricycle-drive robot

Tricycle drive

- front wheel is powered and steerable
- back wheels tag along...



Starting from

α, the robot's steering angle

$V_F = \omega B / \sin(\alpha)$

We conclude

Kinematics

$$\omega = V_F \sin(\alpha) / B$$

$$R = B / \tan(\alpha)$$

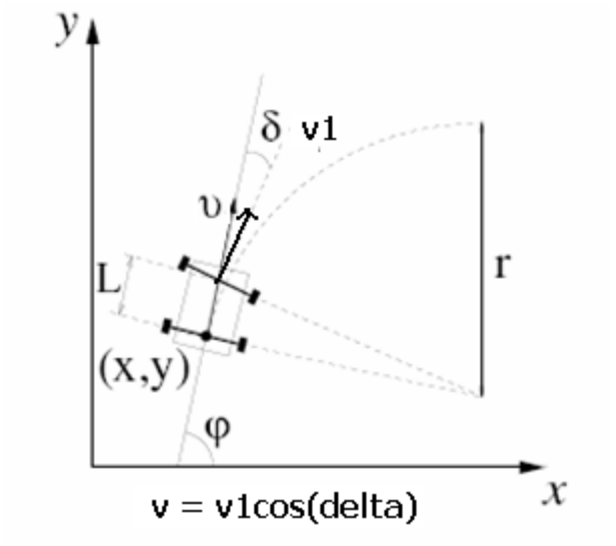
$$V = \omega R = V_F \cos(\alpha)$$

$$V_L = \omega(R+d) = V + dV_F \sin(\alpha) / B$$

$$V_R = \omega(R-d) = V - dV_F \sin(\alpha) / B$$

(velocity only)

Kinematics of the Car



$$\begin{aligned}\dot{x} &= \upsilon\cos\varphi\\ \dot{y} &= \upsilon\sin\varphi\\ \dot{\varphi} &= \frac{\upsilon}{L}\tan\delta.\end{aligned}$$

Integrating above equations you will find the car moves in a circle of radius Lcot(delta)

Delta = 0 corresponds to an infinite turning radius or a straight line,

NOTE:

Since the linear velocity appears in the time evolution of all the state variables, this car model does not have the possibility to make arbitrary rotations while standing still in $\mathbb{R}^2$. It is only able to follow paths that are at least continuously differentiable.

Solution 1: Minimum Reversals Method

1. Let initial state or configuration of the robot be X0. Add this state to a tree T
2. Generate discrete successors for the current state X (a leaf node of T) by varying the steering angle delta in the kinematic equations for the car. Each delta gives a successor node, call it Xn.
3. Among the generated successors Xn (all the leaf nodes of T so far) choose the node with minimum number of reversals and insert it into a heap it is collision free. This step is repeated till a collision free successor node is found
4. Now X = top most node in the heap
5. If X is the goal state or near the goal state STOP else repeat 2 to 4 till goal is reached or heap is empty

Basic RRT Algorithm

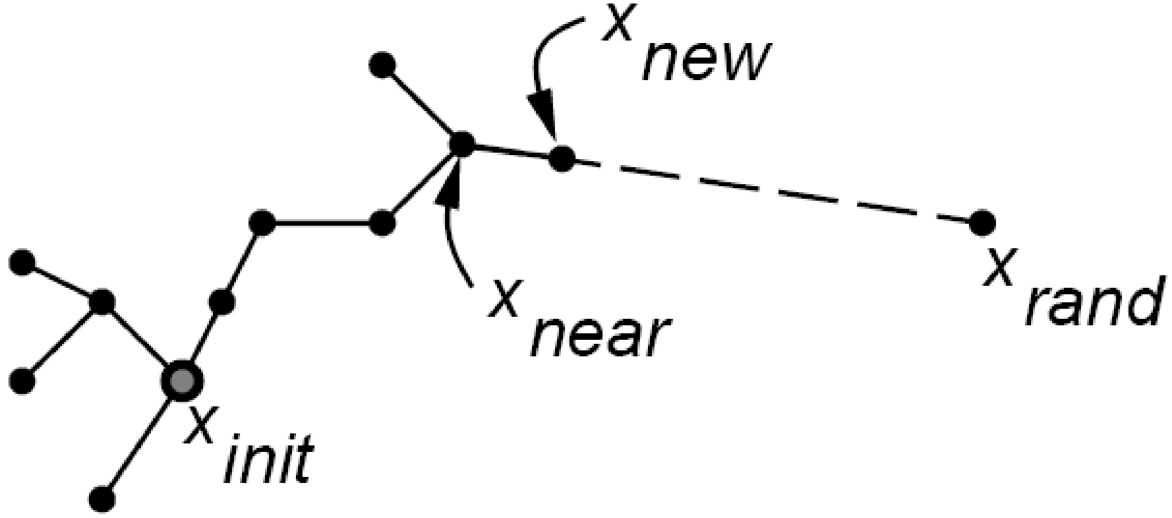
- Suitable for rapidly exploring space
- Inefficient for one-time queries

ISSUES:

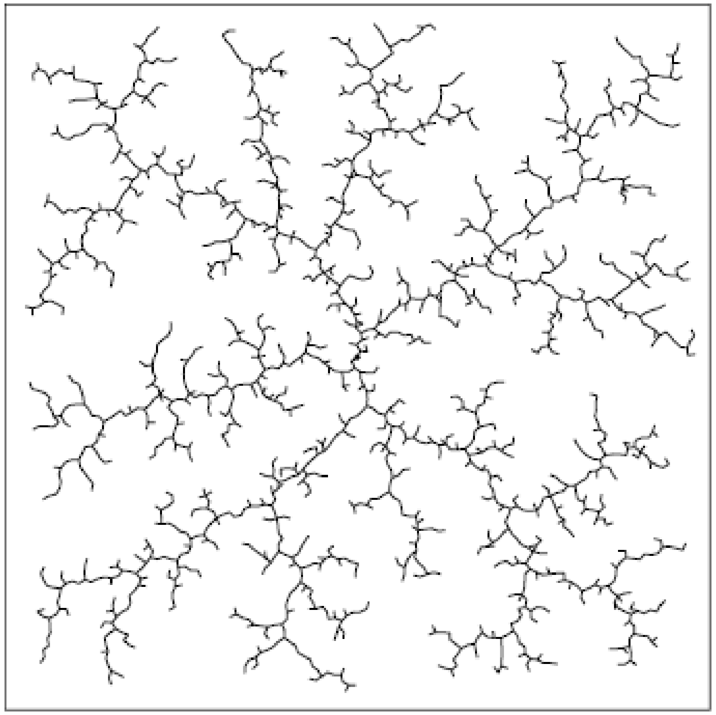
1. Choice of distance metric
2. Choice of thresholds – how much error in final position to allow
3. Choice of data-structure for search

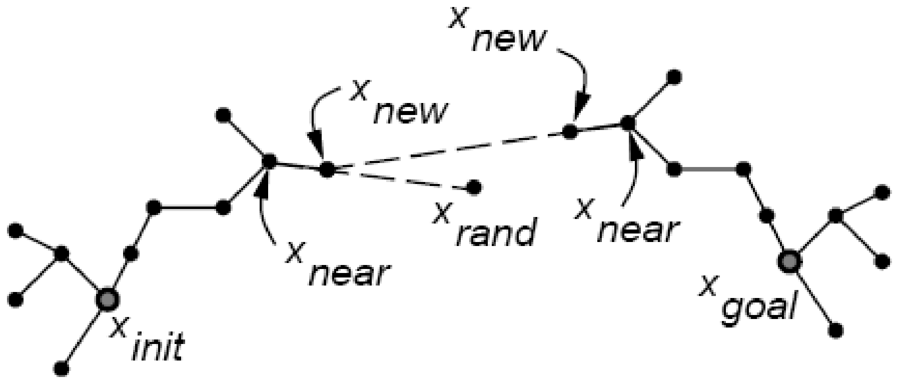
```
RRT-Explore (Tree T, Vertex src)
1: T = {src}
2: for k = 1 to K //(K number of random nodes in workspace W)
3:         $X_k$ = Random-State( )//one such random node
4:         Xn = Nearest-Neighbour(T, X)// finds the node among the leaf nodes that is closest to $X_k$
5:         Xs = SuccessorState(Xn, X, U) //generates a successor to Xn, Xs, that takes Xs closer to $X_k$ on some metric
6:         if( collision-free( Xs ) )
7:         then T = T U { Xs }
```

RRT (illustration)



RRT and Dual RRT (Bidirectional Search)





Dual RRT with bi-directional search

Bidirectional RRT Algorithm

- Efficient for one-time queries
- Multiple rendezvous points
- Faster search

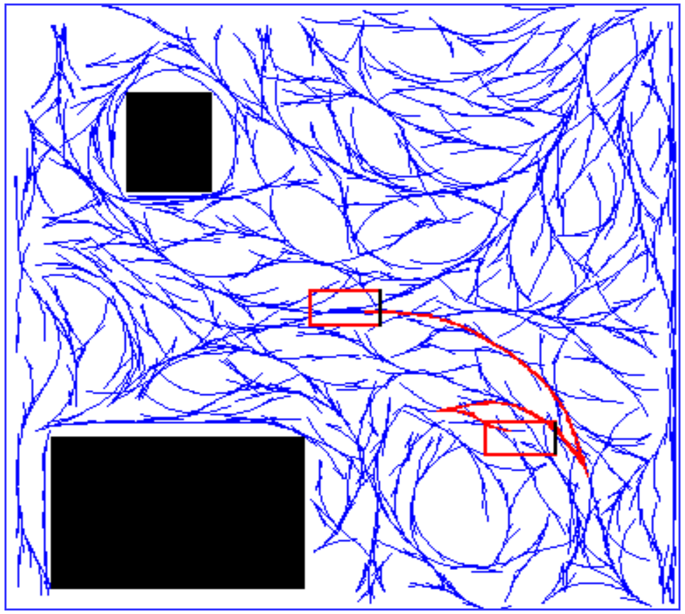
HEURISTICS USED:

Goal-bias – by a coin flip, either a random state or the target state is used.

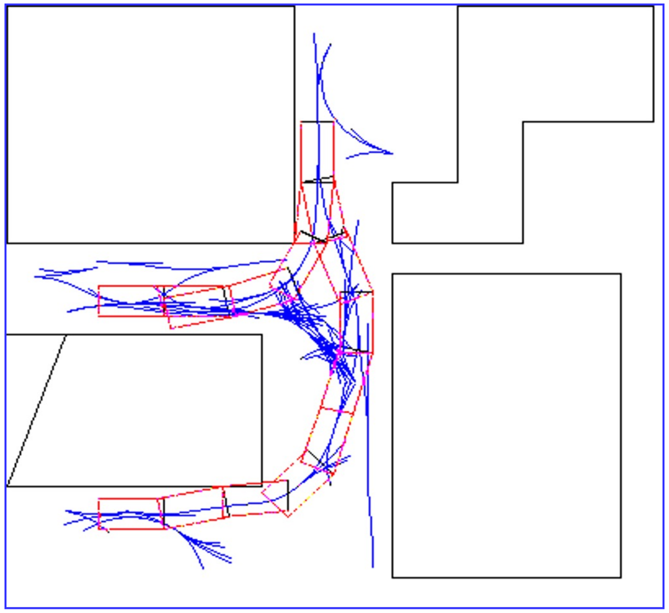
```
RRT-Create (Tree T, State src, State dest)
1:  T1 = {src};  T2 = {dest}
2:  while( true )
3:          X = Random-State( );
4:          if( Extend( T1, X, Xn) != TRAPPED )
5:          then if( Extend(T2, Xn, Xn') == REACHED)
6:                  then break;
7:          swap( T1, T2 );
```

```
Extend( Tree T, State Xr, State &Xs )
1:  Xn = Nearest-Neighbour(T, Xr)
2:  Xs = SuccessorState(Xn, X, U)
3:  if( collision-free( Xs ) )
4:  then T = T U { Xs };
5:  else return TRAPPED;
6:  if( Xs is-close-enough-to Xr )
7:  then  return REACHED;
8:  else return ADVANCED;
```

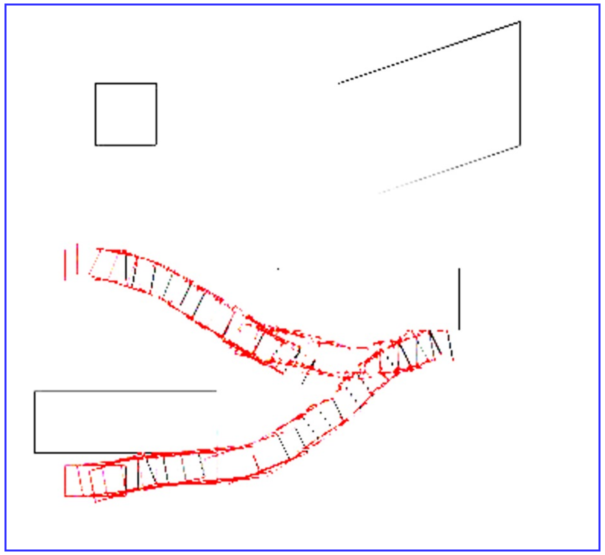
Results with RRTs



Exploring with RRTs

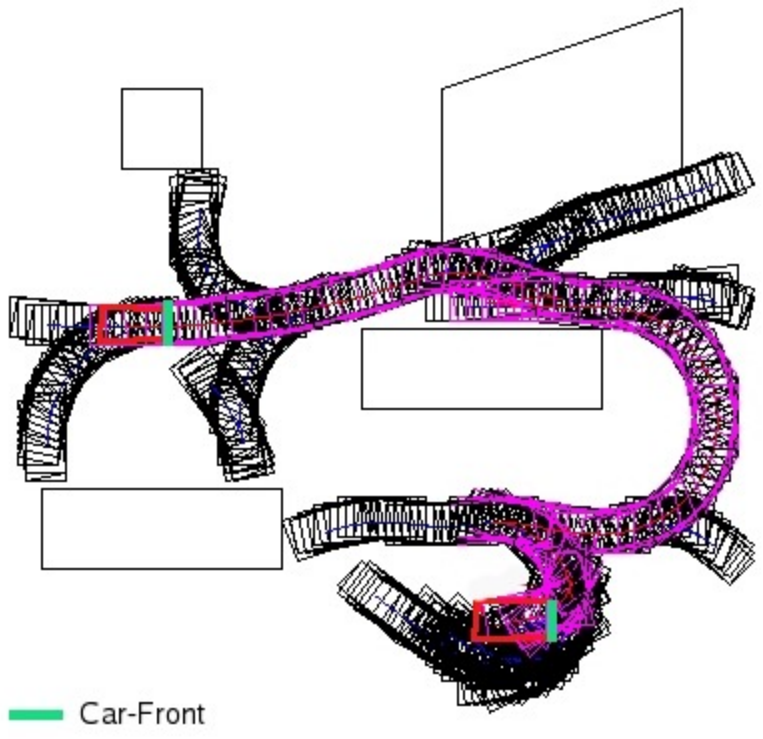


Unbiased RRT search

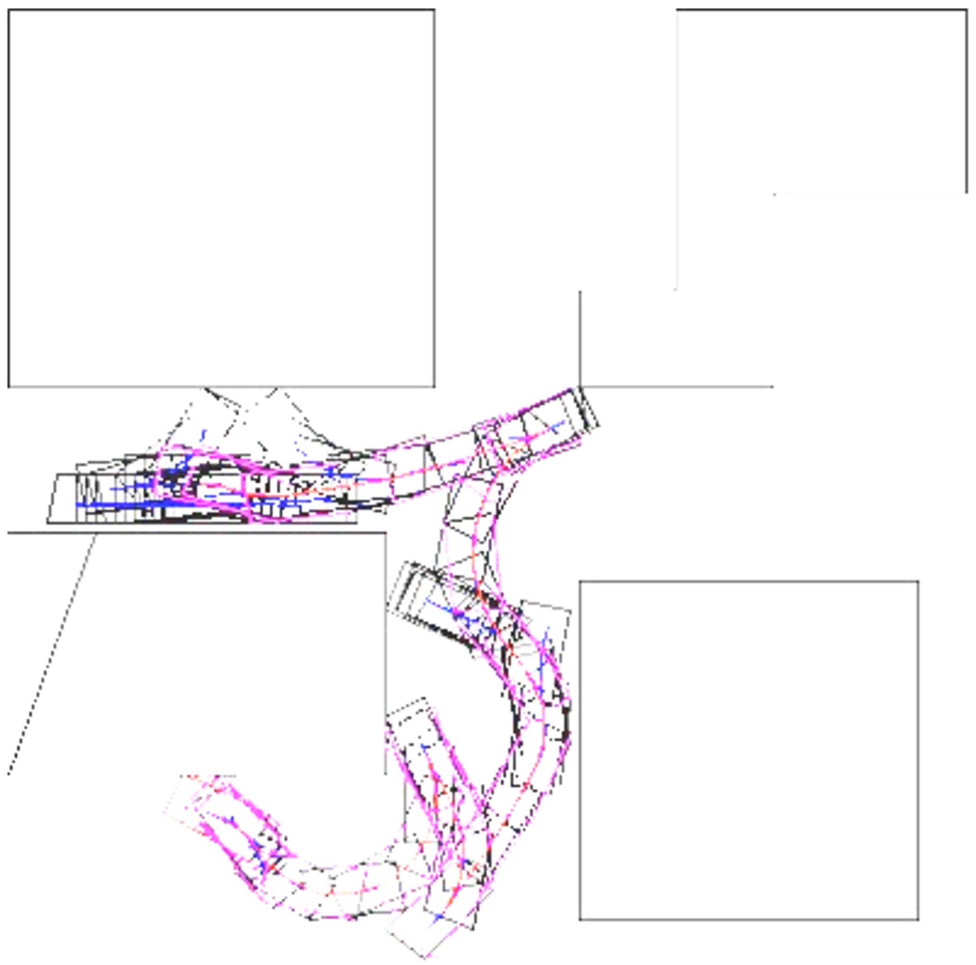


Goal-biased RRT search

Some Results



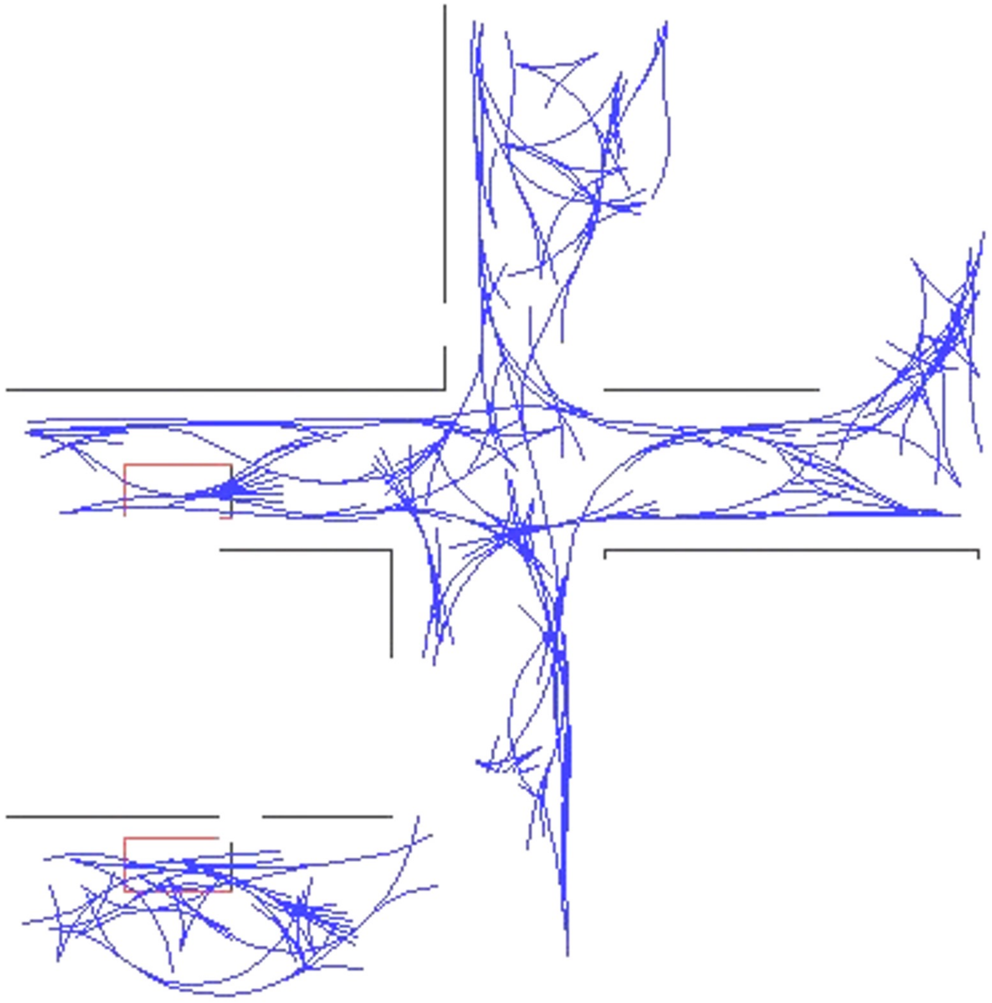
Results with RRTs



Unbiased bi-directional search usually yields such sub-optimal paths as the one shown in the figure.

The black pink represents the actual car path, while he black represents the exploration conducted by the tree

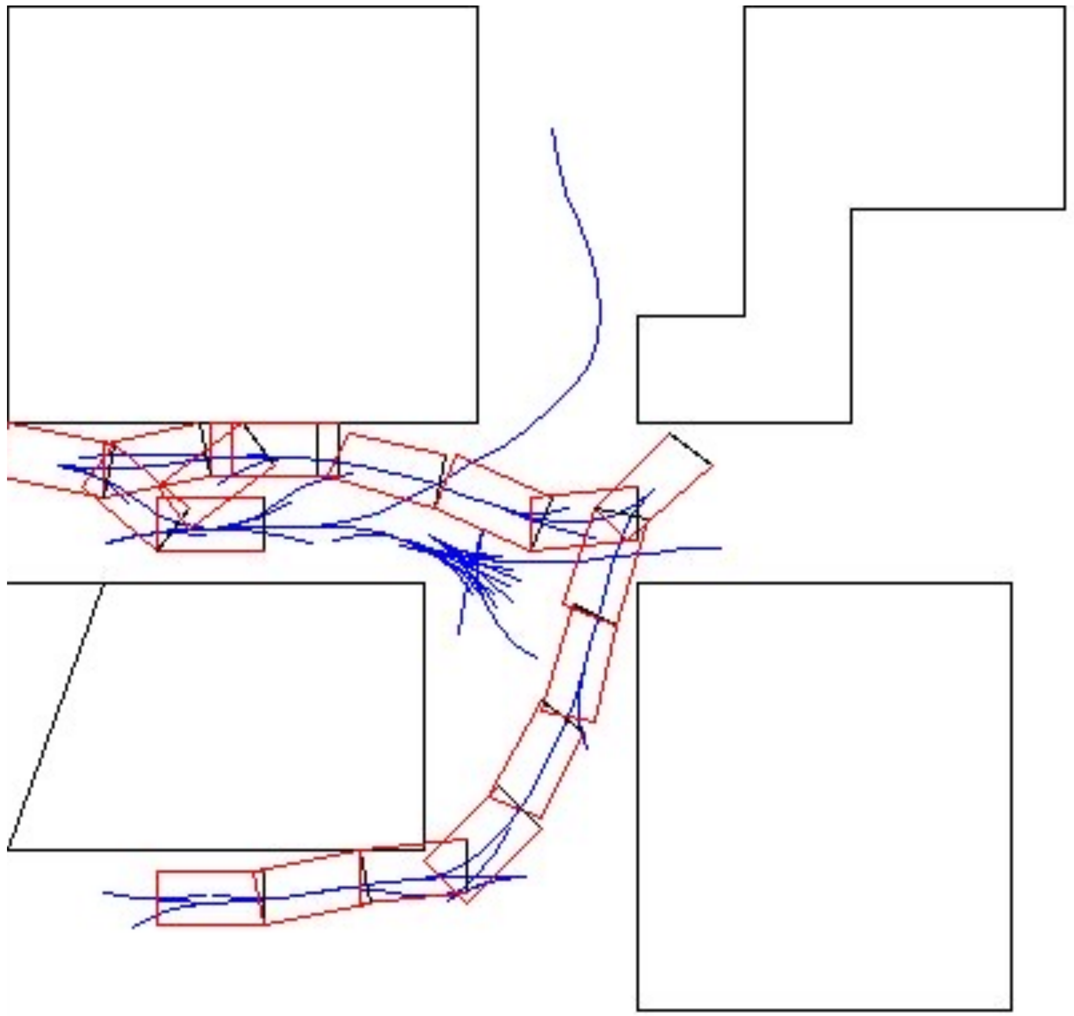
Results with RRTs



Unbiased bi-directional search is great for exploring in the state-space.

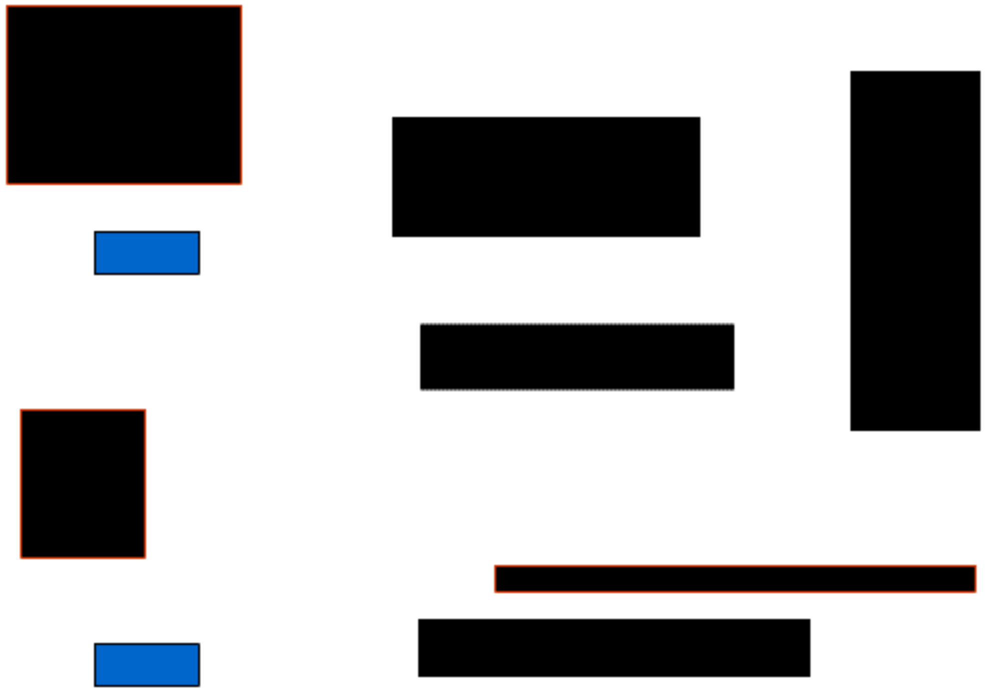
A possible application could be store the pre-explored map as an incomplete planner.

Results with RRTs



Biased (goal-biased) bi-directional search gives good results for difficult cases.

Results with RRTs



Problems with RRT

- The RRT bypasses the need for a local planner. Hence it is always difficult to reach the final state exactly or even nearly exactly through a discrete search
- The RRT uses a metric by which it decides the node closest to the random state. However many a time deciding the metric is equivalent to solving the motion planning problem as such. In other words there is no suitable or appropriate metric often to decide closeness between two states.
- Bidirectional RRT: Find it out☺

The FR SteerBot at iRL

- Has both front and reer wheel steer
- Gives rise to a smaller turning radius (can navigate in tighter spaces)
- Also has the property of parallel steer

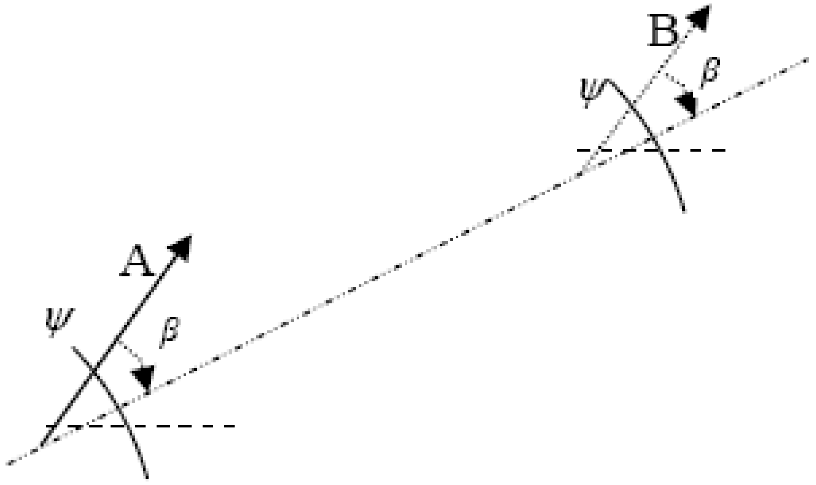
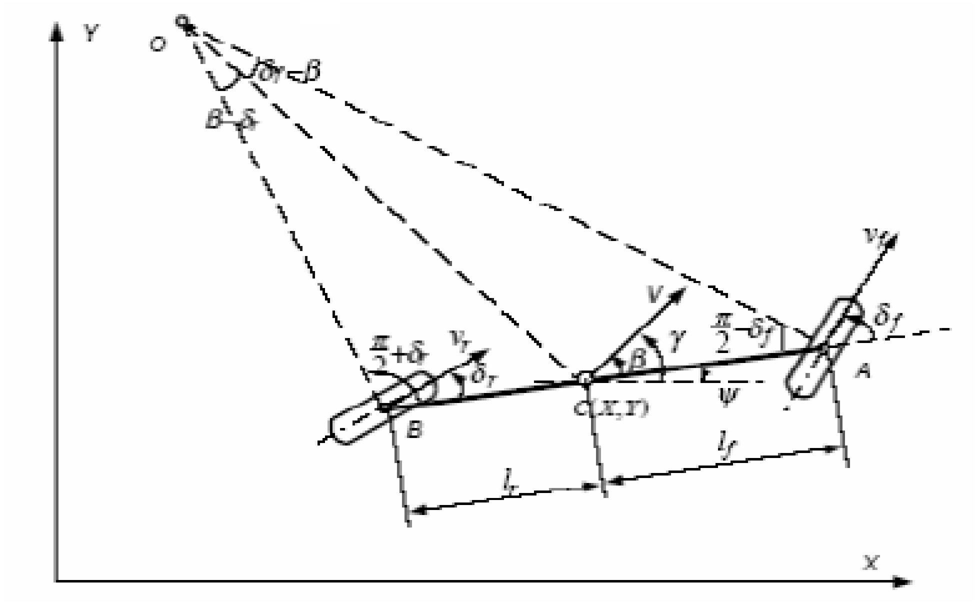


Fig. 2 Parallel steer at β slip

Kinematics of the FR Bot

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} v\cos(\psi+\beta) \\ v\sin(\psi+\beta) \\ \dfrac{v\cos\beta(\tan\delta_f - \tan\delta_r)}{l_f + l_r} \end{bmatrix}$$

$$\frac{l_f + l_r}{\cos\beta\left(\tan\delta_f - \tan\delta_r\right)}$$



The radius of the car is given by $\dfrac{l_f + l_r}{\cos\beta\left(\tan\delta_f - \tan\delta_r\right)}$ centered at O

1). This radius is the distance from O to C. The parallel steer condition occurs when $\delta_f = \delta_r$ resulting in an infinite turn radius. This implies that the car translates without changing its orientation.