Robotics

Lecture 2: Robot Motion

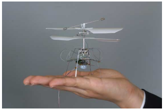
See course website
http://www.doc.ic.ac.uk/~ajd/Robotics/ for up to
date information.

Andrew Davison
Department of Computing
Imperial College London

Robot Motion

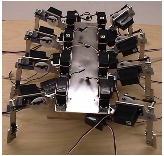
- A mobile robot can *move* and *sense*, and must *process information* to link these two. In this lecture we concentrate on robot movement, or locomotion.
- Robots might want to move in water, in the air, on land, in space...?

AUV



Micro UAV

Zero-G Assistant



Spider

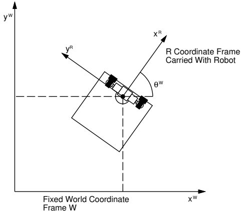
Humanoid

- In this course we will concentrate on wheeled robots which move on fairly flat surfaces.

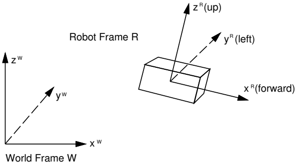
Robot Motion Topics

1. Defining the position of a robot, and the possible ways it can move.
2. Standard wheel configurations for robots, and the motions they allow.
3. Odometry: calculating robot motion from wheel rotations.
4. Actuating and controlling robot wheels.
5. Integrating robot motion on the 2D plane.
6. Position-based path planning.
7. Local and global planning with obstacles.

Motion and Coordinate Frames



2D



3D

- We define two coordinate frames: a *world frame W* anchored in the world, and a *robot frame R* which is carried by and stays fixed relative to the robot at all times.
- Often we are interested in knowing the robot's *location*: i.e. what is the transformation between frames W and R?

Degrees of Motion Freedom

- A rigid body which translates and rotates along a 1D path has 1 degree of freedom (DOF): translational. Example: a train.
- A rigid body which translates and rotates on a 2D plane has 3 DOF: 2 translational, 1 rotational. Example: a ground robot.
- A rigid body which translates and rotates in a 3D volume has 6 DOF: 3 translational, 3 rotational. Example: a flying robot.
- A *holonomic robot* is one which is able to move instantaneously in any direction in the space of its degrees of freedom.
- Otherwise a robot is called *non-holonomic*.

A Holonomic Ground Robot

- Holonomic robots do exist, but need many motors or unusual designs and are often impractical.
- Ground-based holonomic robots can be made using omnidirectional wheels; e.g.
  `https://youtu.be/xUQ4n1FmiXM`

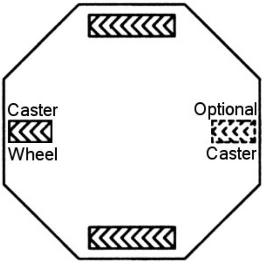
Standard Wheel Configurations

Drive and Steer

Differential Drive

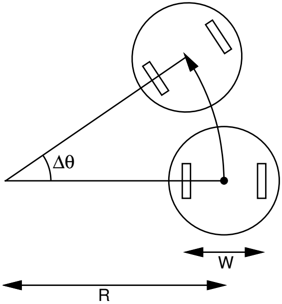
- Simple, reliable, robust mechanisms suitable for robots which essentially move in a plane.
- Both of these configurations are *non-holonomic* (each uses two motors, but has three degrees of movement freedom). For instance, a car-like robot can't instantaneously move sideways.

Differential Drive



- Two driving wheels on the left and right sides of a robot, each driven by its own motor. Steering is achieved by setting different wheel speeds.
- Wheels run at equal speeds for straight-line motion.
- Wheels run at equal and opposite speeds to turn on the spot.
- Other combinations of speeds lead to motion in a circular arc.

Circular Path of a Differential Drive Robot



We define the wheel velocities of the left and right wheels respectively to be v_L and v_R (linear velocities of the wheels over the ground: e.g. $v_L = r_L \omega_L$, where r_L is the radius of the wheel and ω_L is its angular velocity). The width between the wheels of the differential drive robot is W.

- Straight line motion if $v_L = v_R$
- Turns on the spot if $v_L = -v_R$
- More general case: moves in a circular arc.

Circular Path of a Differential Drive Robot

To find radius R of curved path: consider a period of motion Δt where the robot moves along a circular arc through angle $\Delta\theta$.

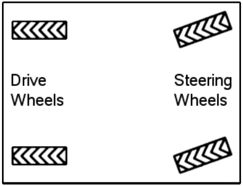
- Left wheel: distance moved $= v_L \Delta t$; radius of arc $= R - \frac{W}{2}$.
- Right wheel: distance moved $= v_R \Delta t$; radius of arc $= R + \frac{W}{2}$.
- Both wheel arcs subtend the same angle $\Delta\theta$ so:

$$\begin{aligned} \Delta\theta &= \frac{v_L \Delta t}{R - \frac{W}{2}} = \frac{v_R \Delta t}{R + \frac{W}{2}} \\ \Rightarrow \frac{W}{2}(v_L + v_R) &= R(v_R - v_L) \end{aligned}$$

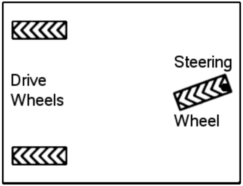
$$\boxed{\Rightarrow R = \frac{W(v_R + v_L)}{2(v_R - v_L)} \qquad \Delta\theta = \frac{(v_R - v_L)\Delta t}{W}}$$

These equations are the basis for *odometry*: given certain control inputs, how does the robot move?

Car/Tricycle/Rack and Pinion Drive



Car

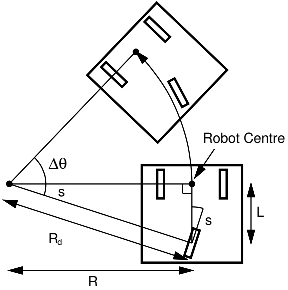


Tricycle

- Two motors: one to drive, one to steer.
- Cannot normally turn on the spot.
- With a fixed speed and steering angle, it will follow a circular path.
- With four wheels, need rear differential and variable ('Ackerman') linkage for steering wheels.

Circular Path of a Car-Like Tricycle Robot

This is a robot configuration which has a single steerable and drivable wheel at the back. The front wheels are free running.



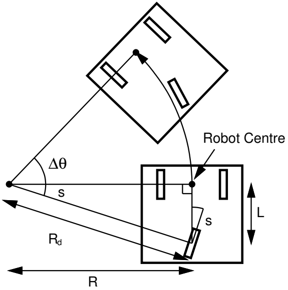
Assuming no sideways wheel slip, we intersect the axes of the front and back wheels to form a right-angle triangle, and obtain:

$$R = \frac{L}{\tan s} \ .$$

The radius of the path that the rear driving wheel moves in is:

$$R_d = \frac{L}{\sin s} \ .$$

Circular Path of a Car-Like Tricycle Robot



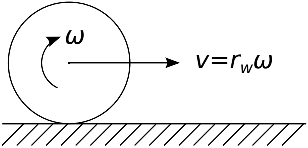
In time Δt the distance along its circular arc moved by the drive wheel is $v\Delta t$, so the angle $\Delta\theta$ through which the robot rotates is:

$$\Delta\theta = \frac{v\Delta t}{R_d} = \frac{v\Delta t \sin s}{L} .$$

$$\boxed{R = \frac{L}{\tan s} \qquad \Delta\theta = \frac{v\Delta t \sin s}{L}}$$

Mapping Wheel Rotation Speed to Velocity

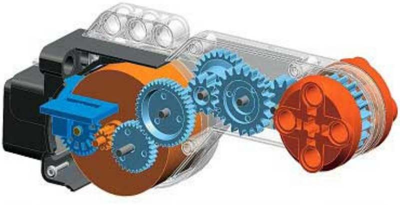
- What is the robot speed, when the wheels turn?



- Calculating a robot's motion based on measuring the angles of motors and how much wheels turn is called *wheel odometry*.
- In principle, we could measure the radius of each wheel r_w to turn angular velocity into linear motion. However, in practice (due to hard to model factors, such as surface slip and tyre softness) it is much better to *calibrate* such things empirically. i.e., via experiments (guided trial and error), work out the scaling between the motor reference angle and distance travelled over the ground.

Actuation of Driving Wheels: DC Motors

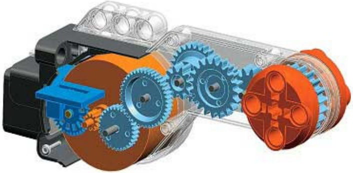
- In practise, how do we make a wheel turn at a desired angular rate?
- DC motors are available in all sizes and types.
- A power signal is sent to the motor (using Pulse Width Modulation PWM).
- For precision, encoders and feedback can be used for *servo* control using a PID control law. Many real motors have built in *encoders* which are sensors which measure their angular position.
- This is an internal view of a Lego NXT motor, which includes an encoder and a gearing system to turn high rotation rate/low torque at the DC motor into low rotation rate/high torque at the output connector.



Powering and Controlling Motors

- Most basically, we can set an amout of power to send to a motor, which will cause it to start to move. Most often, this will be a voltage signal with a fixed amplitude but with the amount of ‘fill-in’ set using Pulse Width Modulation (PWM).
- A set power level will cause an angular response of the motor which depends on various things: the friction in the gears; the amount of load connected to the motor (e.g. the mass of the robot it has to move, or the resistance it is pushing).

Feedback or Servo Control Using an Encoder



- A Lego motor has an encoder which records angular position. (Actually the encoder is attached directly to the motor spindle, but the driver software scales this via the gearing to report the angular position of the orange end effector of the motor).
- So we can use feedback control (servo control) to make the motor do what we want.
- Principle: decide where we want the motor to be at every point in time. At a high rate, check where the motor actually is from the encoder. Record the difference (the error). Send a power demand to the motor depending on the error, aiming to reduce it.
- Our motors: record motion rotational position in degrees.
- Two main modes: position control (where demand is a constant) and velocity control (where demand increases linearly with time).

PID (Proportional/Integral/Differential) Control

- Error $e(t)$ is demand minus actual position.
- PID expression: sets power as a function of error:
  $$P(t) = k_p e(t) + k_i \int_{t_0}^{t} e(\tau) d\tau + k_d \frac{de(t)}{dt}$$
- k_p, k_i and k_d are gain constants which can be tuned.
- k_p is the main term: high values give rapid response but possible oscillation.
- k_i, integral term, can be increased to reduce steady state error.
- k_d, differential term, can be increased to reduce settling time.

Motion and State on a 2D Plane

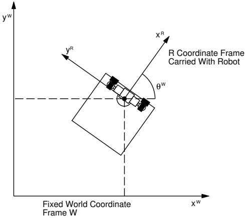
- If we assume that a robot is confined to moving on a plane, its location can be defined with a *state vector* $\mathbf{x}$ consisting of three parameters:

$$\mathbf{x} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix}$$

- x and y specify the location of the pre-defined 'robot centre' point in the world frame.
- θ specifies the rotation angle between the two coordinate frames (the angle between the x^W and x^R axes).
- The two coordinate frame coincide when the robot is at the origin, and $x = y = \theta = 0$.

Integrating Motion in 2D

- 2D motion on a plane: three degrees of positional freedom, represented by (x, y, θ) with $-\pi < \theta <= \pi$.
- Consider a robot which only drives ahead or turns on the spot:



- During a straight-line period of motion of distance D:

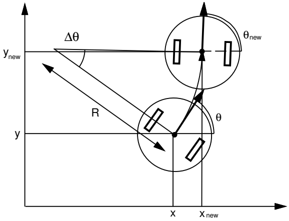
$$\begin{pmatrix} x_{new} \\ y_{new} \\ \theta_{new} \end{pmatrix} = \begin{pmatrix} x + D\cos\theta \\ y + D\sin\theta \\ \theta \end{pmatrix}$$

- During a pure rotation of angle angle α:

$$\begin{pmatrix} x_{new} \\ y_{new} \\ \theta_{new} \end{pmatrix} = \begin{pmatrix} x \\ y \\ \theta + \alpha \end{pmatrix}$$

Integrating Circular Motion Estimates in 2D

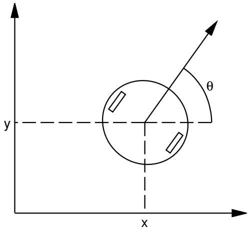
- Simple rotate, move motion like Bigtrak!
  `https://www.youtube.com/watch?v=due9mvuUL-I`.



- More generally, in the cases of both differential drive and the tricycle robot, we were able to obtain expressions for R and $\Delta\theta$ for periods of constant circular motion. Given these:

$$\begin{pmatrix} x_{new} \\ y_{new} \\ \theta_{new} \end{pmatrix} = \begin{pmatrix} x + R(\sin(\Delta\theta + \theta) - \sin\theta) \\ y - R(\cos(\Delta\theta + \theta) - \cos\theta) \\ \theta + \Delta\theta \end{pmatrix}$$

Position-Based Path Planning



Assuming that a robot has *localisation*, and knows where it is relative to a fixed coordinate frame, then position-based path planning enables it to move in a precise way along a sequence of pre-defined waypoints. Paths of various curved shapes could be planned, aiming to optimise criteria such as overall time or power usage. Here we will consider the specific, simple case where we assume that:

- Our robot's movements are composed by straight-line segments separated by turns on the spot.
- The robot aims to minimise total distance travelled, so it always turns immediately to face the next waypoint and drives straight towards it.

Position-Based Path Planning

In one step of path planning, assume that the robot's current pose is (x, y, θ) and the next waypoint to travel to is at (W_x, W_y).

- It must first rotate to point towards the waypoint. The vector direction it must point in is:

$$\begin{pmatrix} d_x \\ d_y \end{pmatrix} = \begin{pmatrix} W_x - x \\ W_y - y \end{pmatrix}$$

  The absolute angular orientation α the robot must drive in is therefore given by:

$$\alpha = \tan^{-1} \frac{d_y}{d_x}$$

Care must be taken to make sure that α is in the correct quadrant of $-\pi < \alpha \leq \pi$. A standard $\tan^{-1}$ function will return a value in the range $-\pi/2 < \alpha <= \pi/2$. This can be also achieved directly with an `atan2(dy, dx)` in most languages.

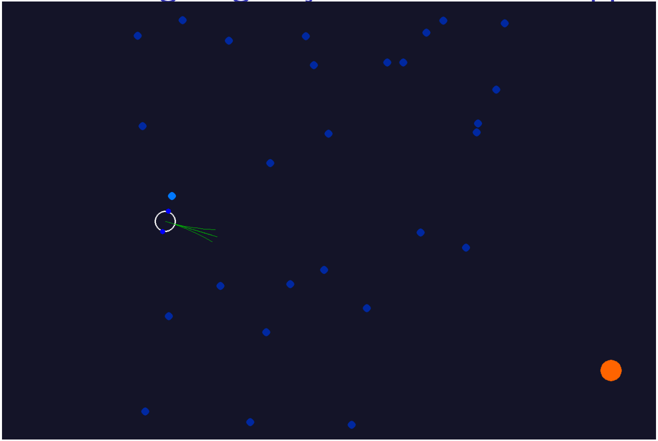
Position-Based Path Planning

- The angle the robot must rotate through is therefore $\beta = \alpha - \theta$. If the robot is to move as efficiently as possible, care should be taken to shift this angle by adding or subtracting 2π so make sure that $-\pi < \beta \leq \pi$.
- The robot should then drive forward in a straight line through distance $d = \sqrt{d_x^2 + d_y^2}$.

More General Planning

- If a robot has a map of environment, how can it make a plan to get from one place to another in the presence of obstacles?
- We have already looked at the specific case of purely position-based planning, where a robot has a sequence of waypoints to follow and it follows straight line paths between them.
- What about more generally, such as if the robot wants to plan a path around a complicated set of obstacles?

Local Planning: e.g. Dynamic Window Approach



- Consider robot dynamics and possible changes in motion it can make within small time dt.
- For each possible motion look ahead longer time τ. Calculate benefit/cost based on distance from target and obstacles.
- Choose the best and execute for dt, then do it again.

Dynamic Window Approach for Differential Drive Robot

- Robot can control v_L, v_R up to maximum values.
- Max acceleration $\times dt$ is the maximum change we can make to either in time dt (e.g. 0.1s).
- 9 possible actions at each step: each of v_L, v_R can either go down, up or stay the same.
- Use the equations from earlier to predict the new position after longer time τ (e.g. 1.0s) for each action. We know the special simple cases for straight line or pure rotation motion well; in the general case the robot moves on a circular arc:

$$\begin{pmatrix} x_{new} \\ y_{new} \\ \theta_{new} \end{pmatrix} = \begin{pmatrix} x + R(\sin(\Delta\theta + \theta) - \sin\theta) \\ y - R(\cos(\Delta\theta + \theta) - \cos\theta) \\ \theta + \Delta\theta \end{pmatrix}$$

`http://www.doc.ic.ac.uk/~ajd/Robotics/RoboticsResources/planning.py`

Dynamic Window Approach for Differential Drive Robot

- For each motion calculate benefit and cost. Benefit will be one weight times the amount the robot would moved closer to its target at (T_x, T_y). e.g.

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

$$B = W_B \times D_F$$

- Subtract from this a cost which is based on the distance the robot would be from the closest obstacle at (O_x, O_y). e.g

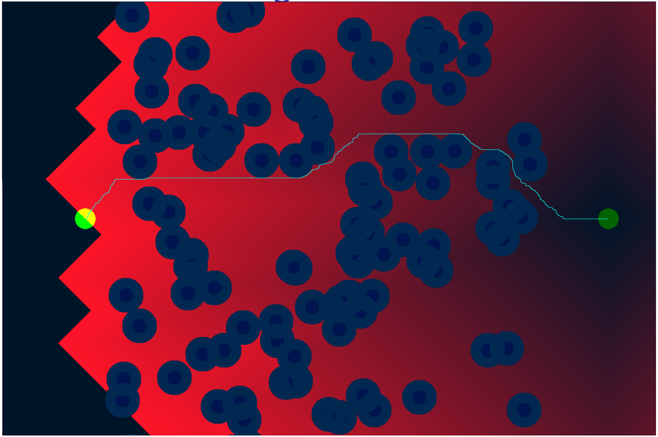
$$C = W_C \times \left(D_{safe} - \left(\sqrt{(O_X - x_{new})^2 + (O_y - y_{new})^2} - r_{robot} - r_{obstacle} \right) \right)$$

  We need to find (O_x, O_y) by search through obstacles. D_{safe} is some distance we would ideally like to stay from contact. r_{robot} and $r_{obstacle}$ are robot and obstacle radii.

- Choose the path with the maximum $B - C$ and execute for dt.

http://www.doc.ic.ac.uk/~ajd/Robotics/RoboticsResources/planning.py

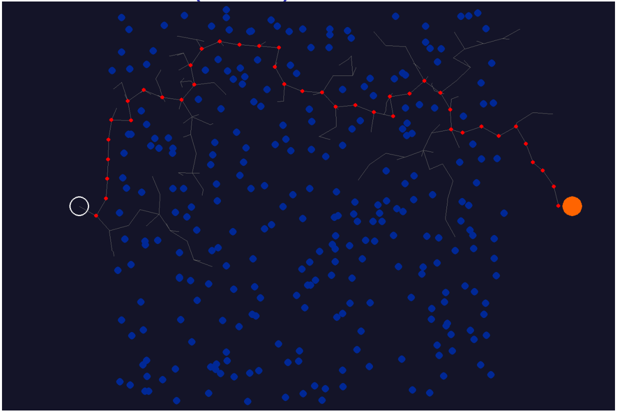
Global Planning: Wavefront Method



- Brute force 'flood fill' breadth first search of whole environment.
- Guaranteed to find shortest route, but slow.
- Thanks a lot to Sajad Saeedi for the implementation!

`http://www.doc.ic.ac.uk/~ajd/Robotics/RoboticsResources/planningwavefront.py`

Global Planning: Rapidly Exploring Randomised Trees (RRT) Method



- Algorithm grows a tree of connected nodes by randomly sampling points and extending the tree a short step from the closest node.
- Expands rapidly into new areas, but without the same guarantees.

`http://www.doc.ic.ac.uk/~ajd/Robotics/RoboticsResources/planningrrt.py`