

# CSCI444/944

## Perception and Planning

Week 6

Motion Control and Programming



# Motion Control and Programming

- Locomotion
- Walking robots
- Mobile robots
- Programming robots

# Locomotion

- Two basic ways of using effectors:
  - to move the robot around => locomotion
  - to move other objects around => manipulation
- This divides robotics into two largely separate categories:
  - mobile robotics
  - manipulator robotics
- Lets look at mobile robots (legs & wheels)

# Locomotion

- There are many kinds of effectors and actuators that can be used to move a robot around.
- The obvious categories are:
  - **legs** (for walking/crawling/climbing/jumping/hopping)
  - **wheels** (for rolling)
  - **flippers** (for swimming)
  - **Propellers & jets** (for flying)
  - **arms** (for swinging/crawling/climbing)
  - ...
- While most animals use legs to get around, legged locomotion is a difficult robotic problem, especially when compared to wheeled locomotion.

# Locomotion

- A robot needs to be stable (i.e., not *wobble* and *fall over* easily).
- There are two kinds of stability:
  - **static** and
  - **dynamic**.
- A ***statically stable*** robot can stand still without falling over.
- This is a useful feature, but can be difficult to achieve:
  - it requires that there be **enough legs/wheels** on the robot to provide sufficient static points of support.

# Locomotion

- For example, people are *not* statically stable.
- In order to stand up, which appears effortless to us, we are actually using active control of our balance.
- Achieved through nerves and muscles and tendons.
- This balancing is largely unconscious:
  - it must be learned or instinctive,
  - requires feedback from sensors,
  - can result in damage or injury if balance is lost.

# Legs vs Wheels

- Wheels are more efficient than legs.
- They also do appear in nature in certain bacteria
  - So the common myth that biology cannot make wheels is not well founded.
- However, evolution favors lateral symmetry and legs are easier to construct and maintain.
- If you look at population sizes, insects are the most populous animals, and have many different leg configurations and locomotion systems.

# Why have Legs?

- Better handling of rough terrain.
  - Only about 1/2 of the world's land mass is accessible by artificial vehicles.
- Use of isolated footholds can provide added support and traction.
  - e.g. a ladder.
- Active suspension
  - decouples path of body from path of feet
  - payload is free to travel despite terrain.



# Why have Legs?

- Less energy loss
- Potentially less weight
- Can traverse more rugged terrain
- Legs do less damage to terrain (environmentally conscious)
- Potentially more maneuverability

# Problems to solve.

- Legged robots have many actuators and degrees of freedom (DOF)
- For example, a typical hexapod has 18-servos (3 for each leg)
- The geometry require both analysis of the robots statics and kinematics.

# Stability of standing and walking

- A ***basic assumption*** of the static gait (statically stable gait) is that the weight of a leg is negligible compared to that of the body,
  - so that the total center of gravity (COG) of the robot is not affected by the leg swing.
- Based on this assumption, the conventional static gait is designed so as to maintain the Center of gravity (COG) of the robot inside of the support polygon.
- This polygon is outlined by each support leg's tip position.

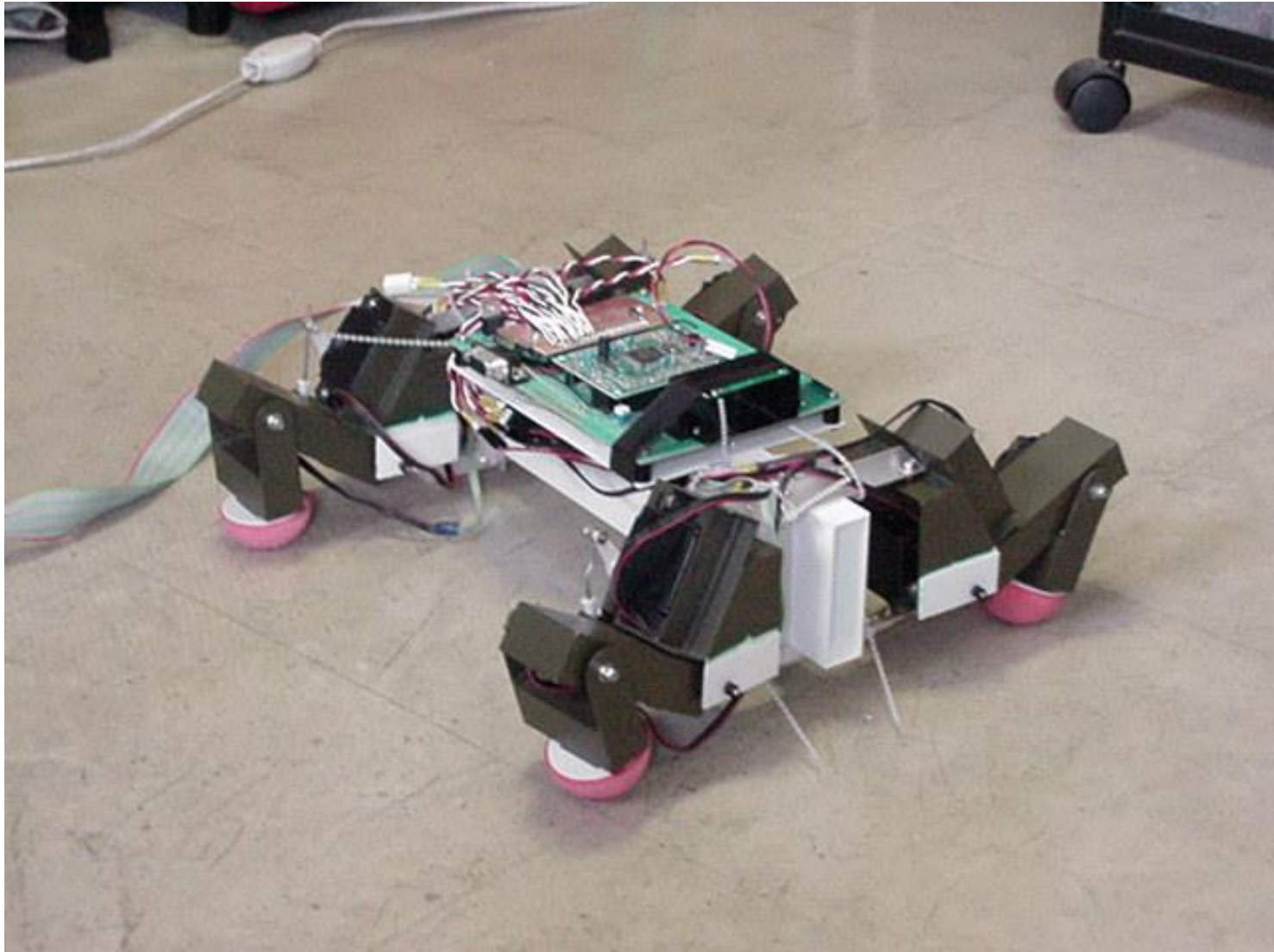
# Stability of standing and walking

- *With more legs, static stability becomes quite simple.*
- In order to remain stable, *the robot's COG must fall under its polygon of support.*
- This polygon is basically the projection between all of its support points onto the surface.
- So in a **two-legged robot**, the polygon is really a line.
- Thus the center of gravity cannot be aligned in a stable way with a point on that line to keep the robot upright.
- Consider a three-legged robot:
  - with its legs in *a tripod organization*, and its body above,
  - Such robot produces a stable polygon of support, thus it is statically stable.

# Stability of standing and walking

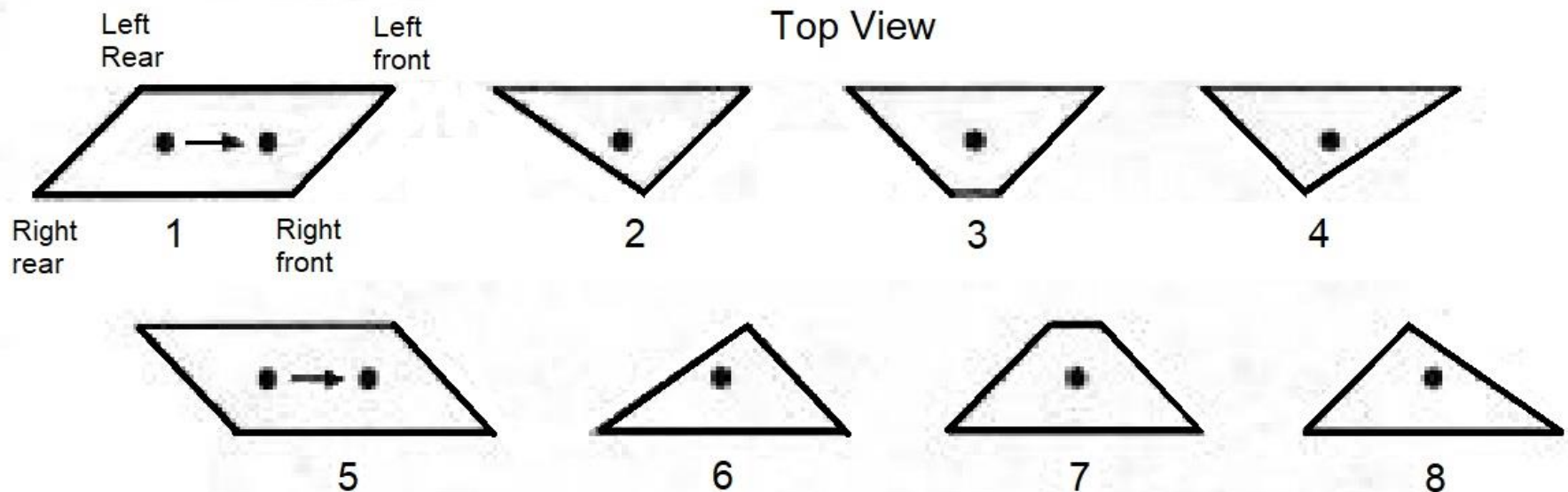
- But what happens when a statically stable robot lifts a leg and tries to move?
- Does its center of gravity stay within the polygon of support?
- It may or may not, depending on the geometry.
- For certain *robot geometries*, it is possible (with various numbers of legs) to always stay statically stable while walking.
- This is very safe, but it is also very slow and energy inefficient.

# Stability of standing and walking



# Stability of standing and walking

- Below shows the sequence of support patterns provided by the feet of a quadruped walking robot.
- Body and legs move to keep the projection of the center of gravity (COG) within the polygon defined by the feet.
- Each vertex is a support foot.
- The dot is the projection of the COG.



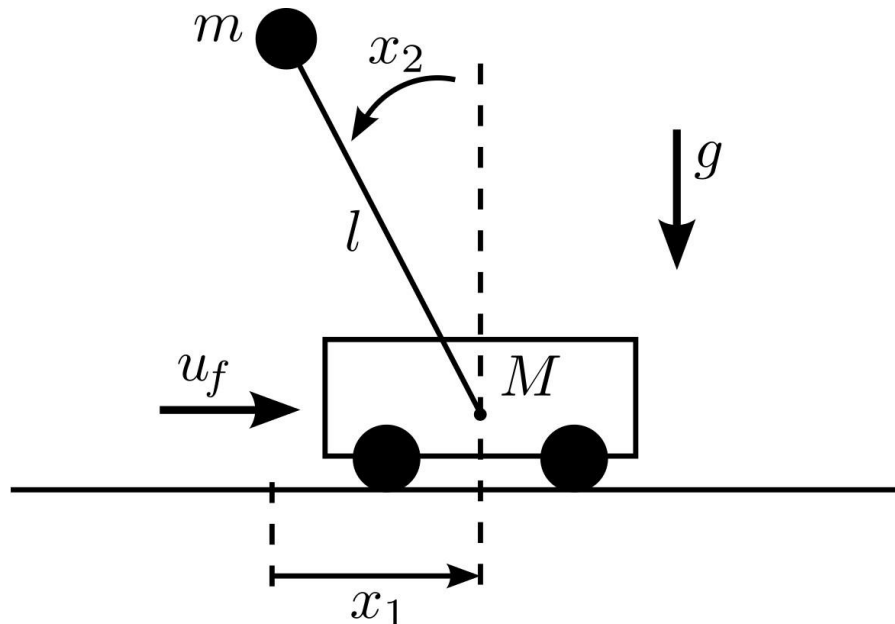
# Stability of standing and walking

- The alternative to static stability is ***dynamic stability*** which allows a robot (or animal) to be stable while moving.
- For example, one-legged hopping robots are dynamically stable:
  - they can hop in place to various destinations and not fall over.
- But **they cannot stop and stay standing**
  - (this is an *inverse pendulum* balancing problem).



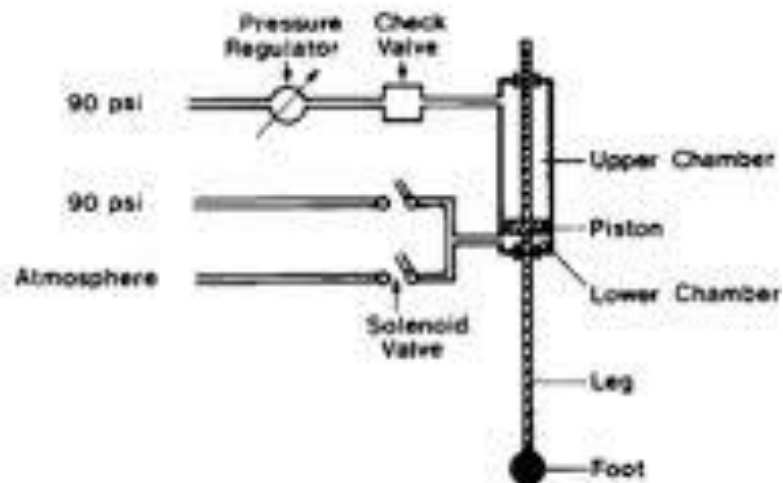
# Active (dynamic) Stability

- Inverted pendulum balanced on cart.
- Only one input, the force driving the cart horizontally, is available for control.



# A Stable Hopping Leg

- Robert Ringrose of MIT AAI97.
- Hopper robot leg stands on its own,
- hops up and down,
- maintaining its balance and correcting it.
- forward, backward left, right, etc., by changing its center of gravity.
- Example: <https://www.youtube.com/watch?v=XFXj81mvInc>



# Stability of standing and walking

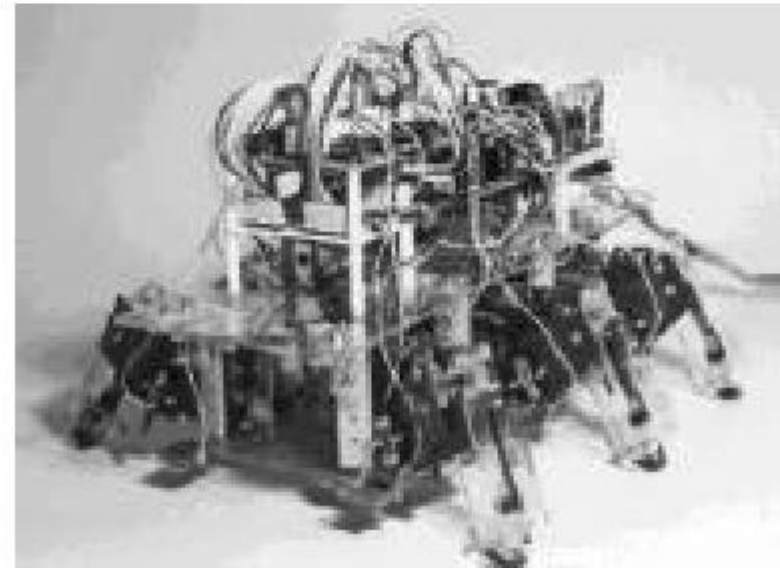
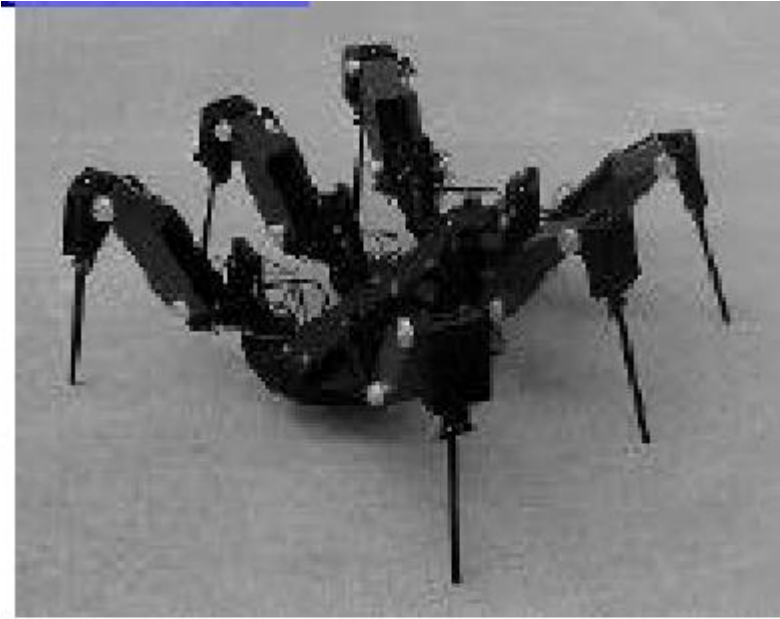
- A statically stable robot can:
  - 1. use dynamically-stable walking patterns – (hard, but fast)
  - 2. use statically stable walking – (easy, but slow).
- A simple way to think about this is by *how many legs are up in the air* during the robot's movement (i.e., gait):
  - 6 legs (**hexapod**) is the most popular number as they allow for a very stable walking gait, the **tripod gait** .
  - if the same three legs move at a time, this is called the **alternating tripod gait**.
  - if the legs vary, it is called the **ripple gait**.

# Insect walking

- Statically stable walking is very *energy inefficient*.
- As an alternative, dynamic stability enables a robot to stay up while moving.
- This requires active control (i.e., the inverse pendulum problem).
- Dynamic stability can allow for greater speed, but requires harder control.
- Balance and stability are difficult problems in control and robotics.
- Thus, when you look at most existing robots, they will have wheels or multiple legs (such as 6).

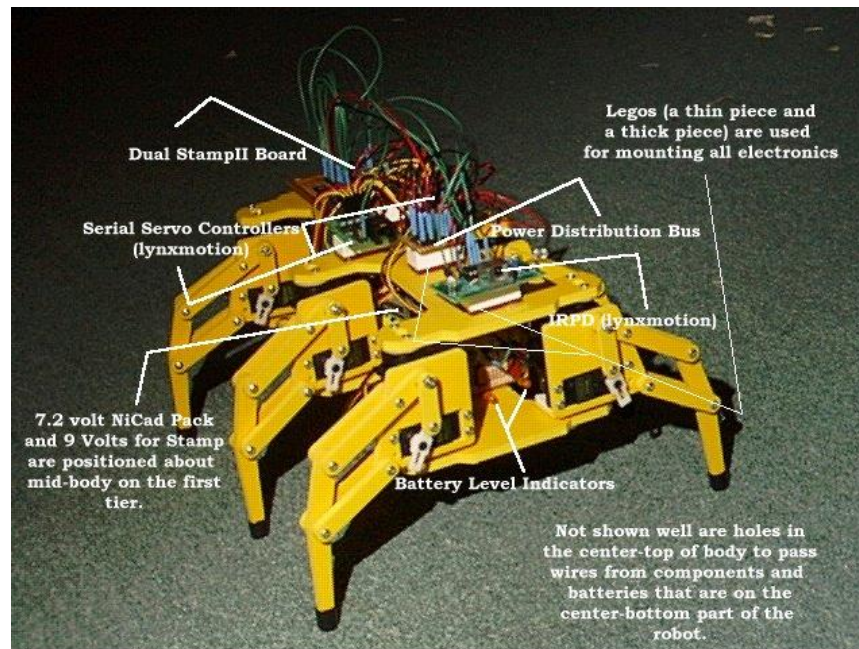
# Hexapods

- Biologically inspired
  - insects
- Potentially very stable as the motion of one leg usually does not affect vehicle stance.
- Relatively simple to come up with a control algorithm
- Examples:
  - <https://www.youtube.com/watch?v=To2Y6Mhu-CE>
  - [https://www.youtube.com/watch?v=W9DOG47\\_xJk](https://www.youtube.com/watch?v=W9DOG47_xJk)



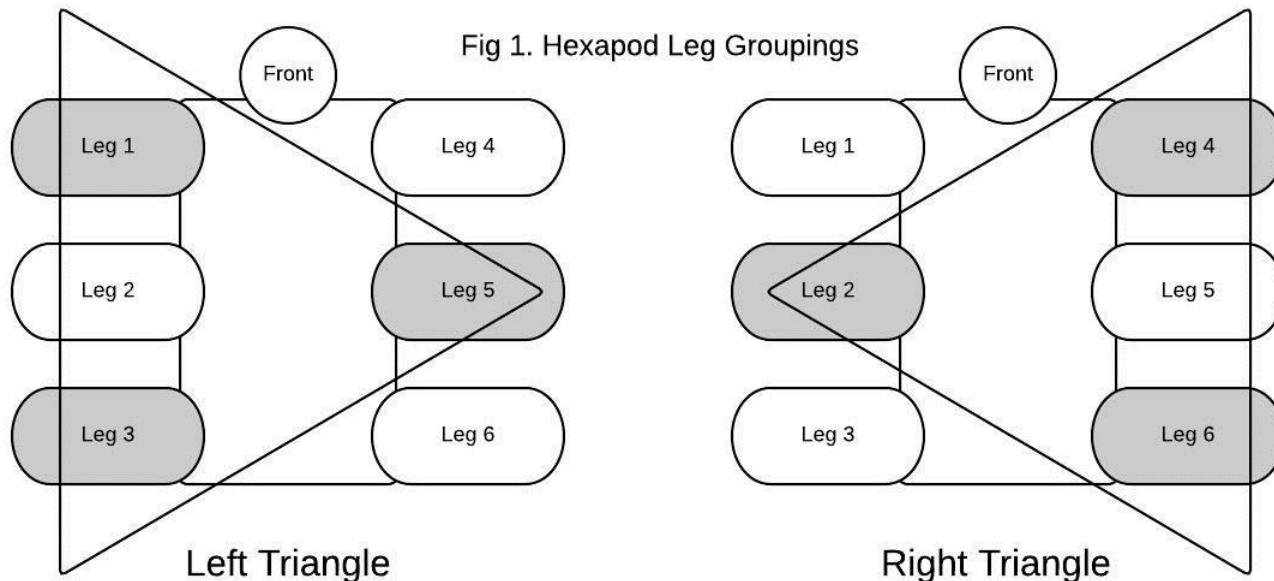
# Hexapod walking

- A rectangular 6-legged robot can lift three legs at a time to move forward, and still retain static stability.
- How does it do that?
- It uses the so-called ***alternating tripod gait***, a biologically common walking pattern for 6 or more legs.



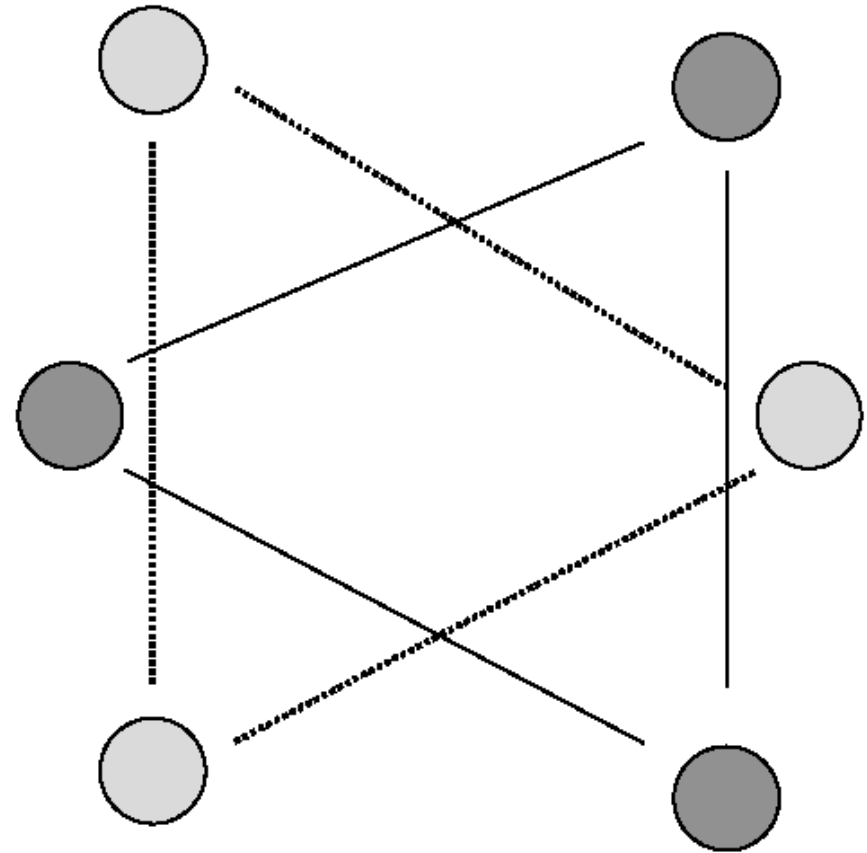
# Hexapod walking

- Characteristic of ***alternating tripod gait***:
  - one middle leg on one side and two non-adjacent legs on the other side of the body lift and move forward at the same time,
  - the other 3 legs remain on the ground and keep the robot statically stable.



# Hexapod walking

- Walking gaits were first reported by D.M. Wilson in 1966.
- A common gait is the “alternating tripod gait”.
- Commonly used by certain insects while moving slowly.





# Hexapod walking algorithm

## Step 1

- legs 1,4,and 5 down, legs 2,3 and 6 up.

## Step 2

- rotate torso 7 and 9 counter-clockwise, torso 8 clockwise.

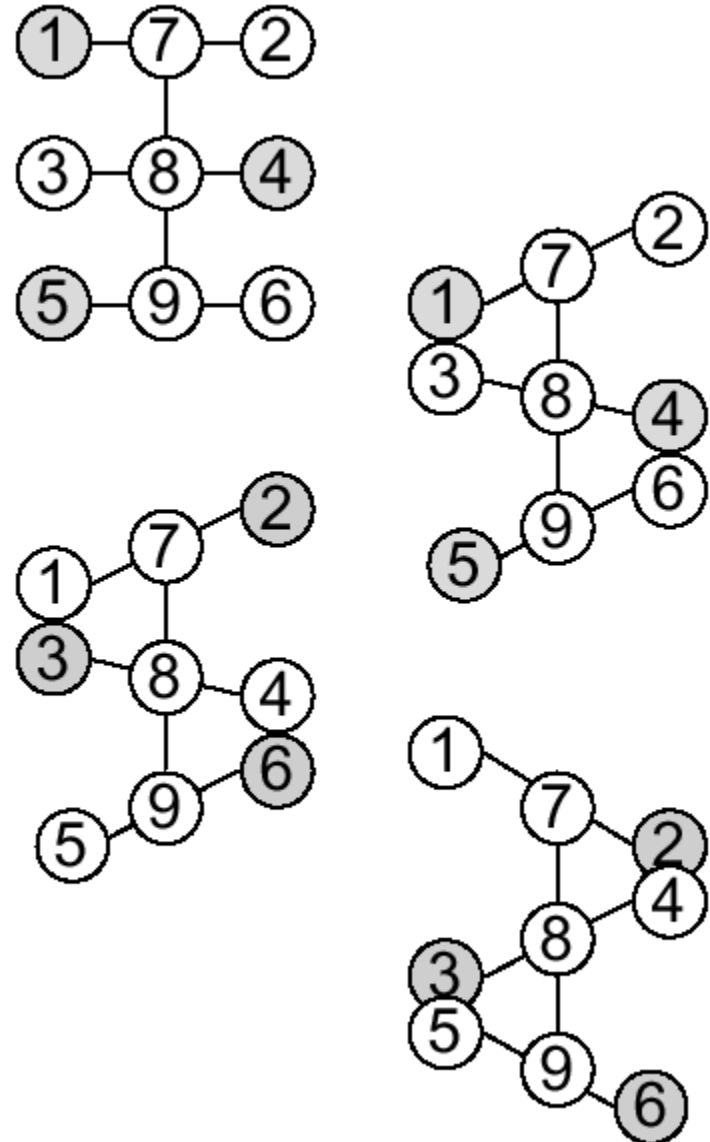
## Step 3

- legs 1,4 and 5 up,
- legs 2,3, and 6 down.

## Step 4

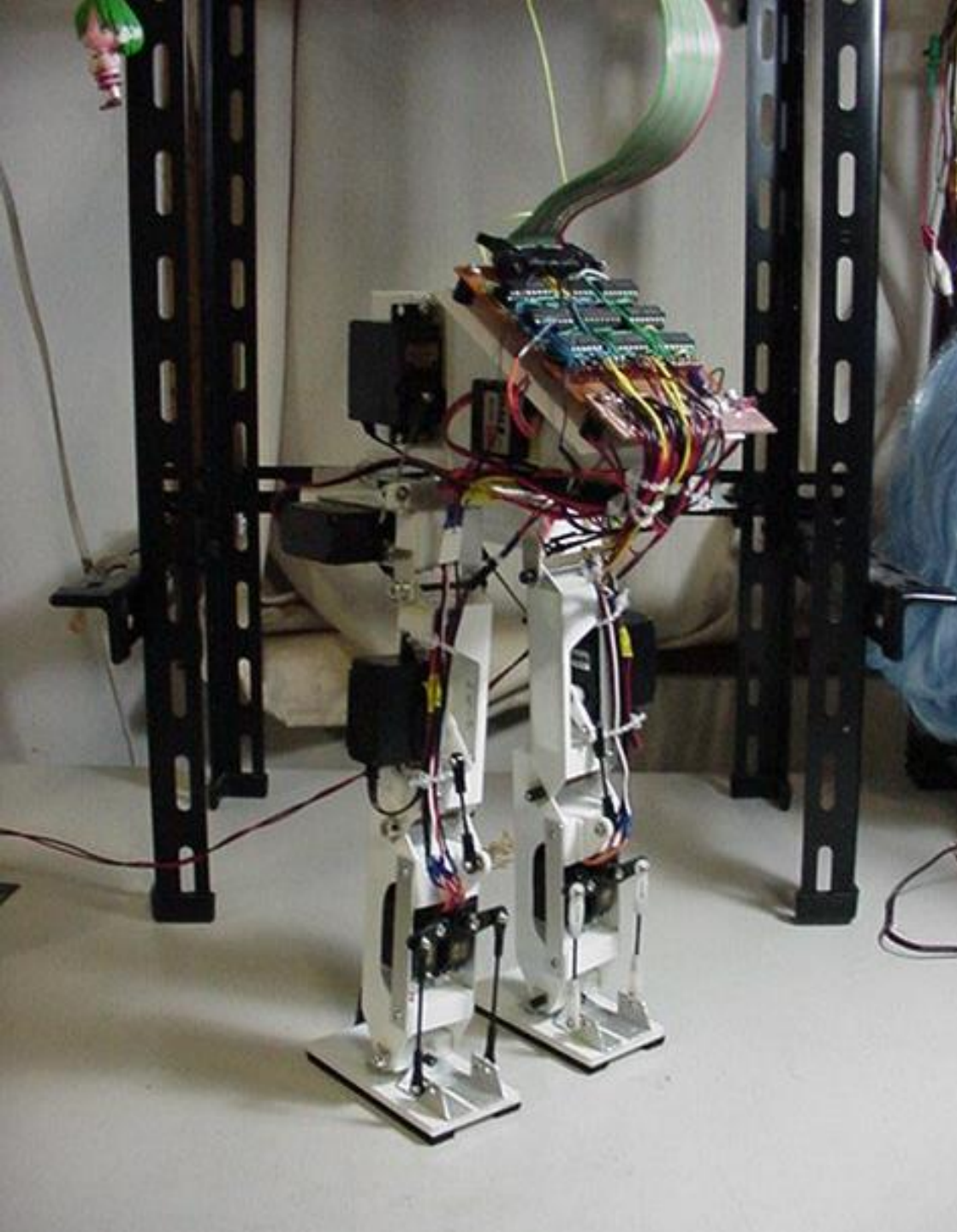
- rotate torso 7 and 9 clockwise, torso 8 counter-clockwise.

Goto step 1



# Other types of insect walking

- **Roaches** move using an alternating gait, and can do so very quickly.
- **Insects** with more than 6 legs (e.g., centipedes and millipedes), use the ripple gait.
  - However, when these insects run really fast, they switch gaits to actually become airborne (and thus not statically stable) for brief periods of time.



# Biped Robots

# Design Methodologies

- Two main design methods
- Biomechanical control
  - Traditional control method
  - Robust and versatile
  - ‘Robotic’ looking gait
  - Difficult and expensive to implement
  - Unnecessarily complex
  - Inefficient and heavy
- Control based on *gait synthesis*



“Asimo X2 at Robodex 2003”  
<http://www.plyojump.com/asimo.html>

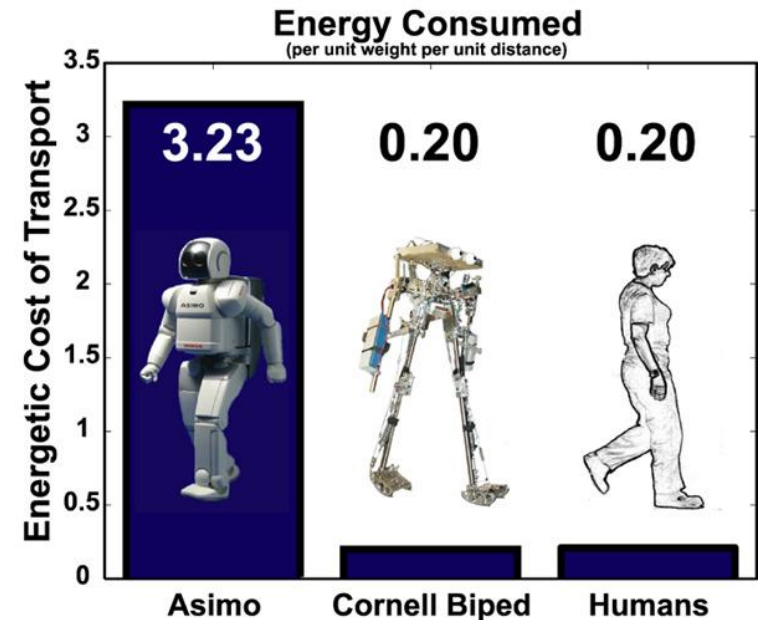
# Control by Gait Synthesis

- Simulate natural kinematics of walking
  - Begins with essentials of walking
  - Actuate only when required
- Relatively new approach
  - McGeer 1990, Wisse 2004
- Inherent sequential design
- Suited to muscle actuation
- Simple control required
- Very efficient
- Watch:

<https://www.youtube.com/watch?v=rJ56d1UTIKQ&t=44s>

vs.

<https://www.youtube.com/watch?v=kzgk4Rnpqc>



Collins et al 2005

# Biped Robots

- Challenges in bipedal robots
- Bipedal are Hyper DOF systems ( $>20$ ) - Complex Kinematics and Dynamics.
- Complex real-time control architecture.
- Complexity limits the trajectory tracking of ease.
- Conventional control algorithms for humanoid robots can run into some problems related to :
  - Mathematical tractability
  - Optimisation
  - Limited extendibility
  - Limited biological plausibility

# Biped Robots

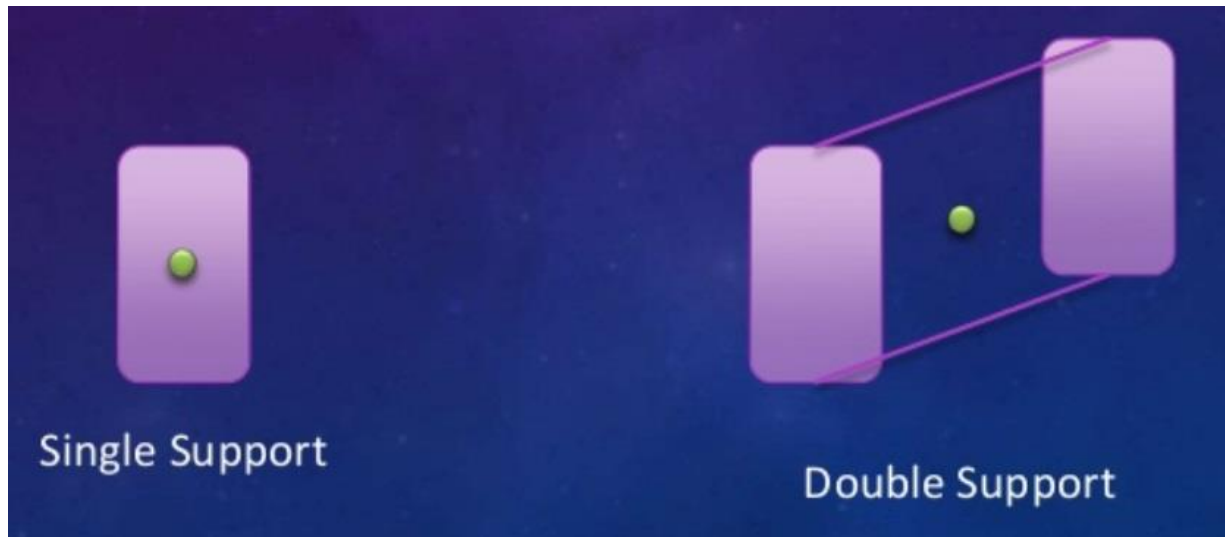
- Biped dynamics and control strategies:



# Biped Robots

## STATIC WALKING:

- In static walking, the biped has to move slowly so that the dynamics can be ignored.
- The biped's projected center of gravity (PCOG) must be within the supporting area.





# Biped Robots

## DYNAMIC WALKING:

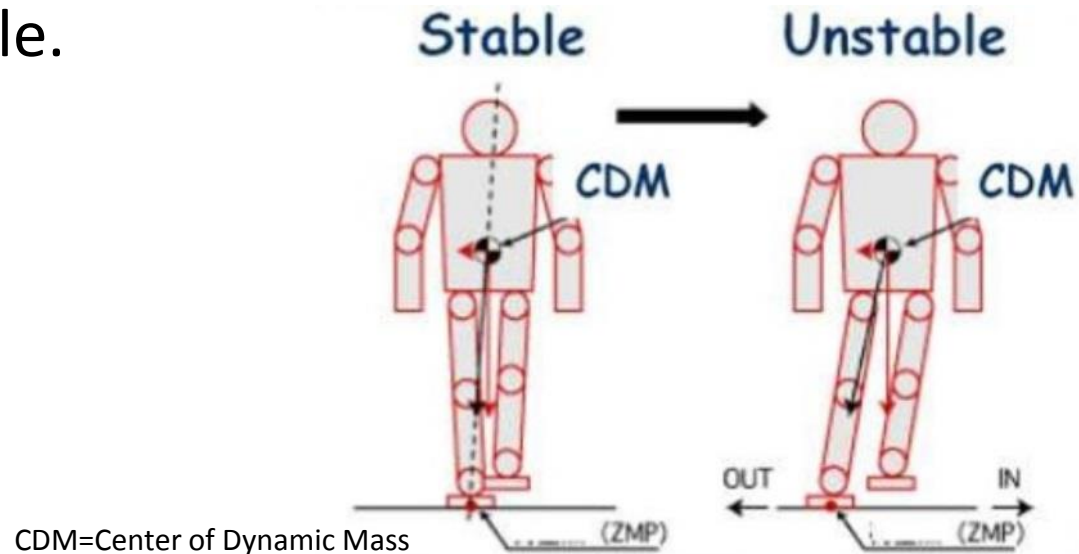
- In dynamic walking, the motion is fast and hence the dynamics cannot be negligible.
- In dynamic walking, we should look at the zero moment point (ZMP) rather than COG.
- The stability margin of dynamic walking is much harder to quantify.

$$\text{Minimize} \quad \int_{t_i}^{t_f} \left\| P_{zmp}(t) - P_{zmp}^d(t) \right\|^2 dt$$

# Biped Robots

## ZERO MOMENT POINT (ZMD):

- ZMP specifies the point with respect to which dynamic reaction force at the contact of the foot with the ground does not produce any moment.
- The point where total inertia force equals zero.
- ZMP is the indicator of the stability of the robot:
- if it is in the foot shadow – stable,
- if not – unstable.



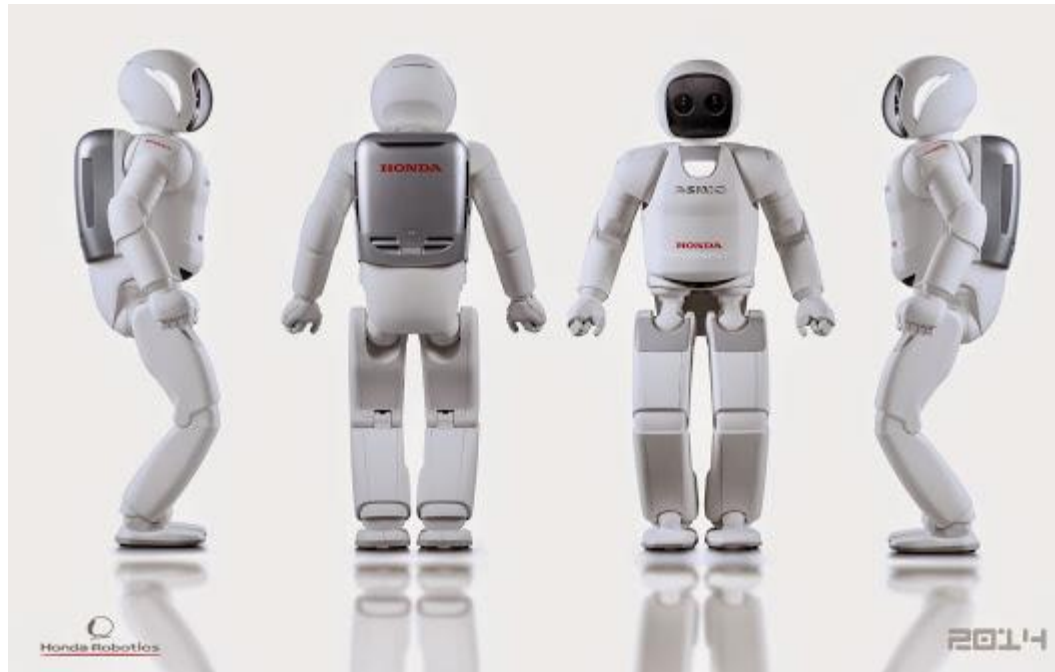
# Biped Robots

Biped control methodologies:

- Model Based
  - Passive Walker
  - Inverse Pendulum
- Biped Dynamics Based
  - Dynamic Stabilization
  - Whole Body Cooperative
- Dynamics Modelless Softcomputing Based
  - Connectionist Theory (ANN)
  - Fuzzy Logic
  - Genetic Algorithms
- Hybrid Intelligent Control
  - ANFISRF
  - Learning and GA Optimized
  - Trajectory Panning.

# Honda ASIMO

- ASIMO is a humanoid robot created in 2000 by Honda.
- ASIMO stands for Advanced Step in Innovative Mobility
  - 11th in line of successive bipedal humanoid model's by Honda.
- It is the 4th man like humanoid robot.



# Honda ASIMO

- Specifications:
  - Weight: 52 kilograms
  - Running Speed: 6 km/h
  - Walking speed: 2.7 km/h
  - Walking speed while carrying objects: 1.6 km/h
  - Height: 130 cm
  - Width: 45 cm
  - Depth: 44 cm
  - Continuous operating time: 40 min – 1 hr.
  - Degrees of Freedom: 34

# Honda ASIMO

- Honda's Dynamic Stability Controller
  - Keep foot flat on the ground (fully actuated)
  - Estimate danger of foot roll by measuring ground reaction forces
  - Carefully design desired trajectories via optimization
  - Keep knees bent (avoid singularity)
  - Adaptive trajectory tracking control (high feedback gains)

# Honda ASIMO

- Biped walking - further challenges
  - Can't compete with humans in terms of:
    - Speed (0.44 m/s top speed)
    - Efficiency (uses roughly 20x as much energy per unit weight, per distance moved)
    - Robustness (Can't yet perform fast actions as humans)
    - DC electric motor runs Hot

# Wheeled Robots

- Wheels are the locomotion effector of choice.
- **Wheeled robots** (as well as almost all wheeled mechanical devices, such as cars) are often built to be statically stable.
- It is important to remember that wheels can be constructed with as much variety and innovative flair as legs:
  - wheels can vary in size and shape,
  - can consist of simple tires,
  - or complex tire patterns,
  - or tracks,
  - or wheels within cylinders within other wheels spinning in different directions to provide different types of locomotion properties.
- So *wheels need not be simple*, but typically they are, because even simple wheels are quite efficient.





# Wheeled Robots

Seekur Jr robot

# Wheeled Robots

- Wheels are more efficient than legs
- Having wheels does not imply *holonomicity*.
- 2 or 4-wheeled robots are usually not **holonomic**.
- A popular and efficient design involves two **differentially-steerable wheels** and a **passive caster**.

**Note:** A mobile robot is holonomic when the number of controllable Degrees Of Freedom (DOF) is equal to the total number of DOF on a robot.

# Wheeled Robots

## Differential steering:

- Two (or more) wheels can be rotated (or steered) separately (individually) and thus differently.
- If one wheel can turn in one direction and the other in the opposite direction, the robot can spin in place.
- This is very helpful for following arbitrary trajectories and escaping dead ends.
- ***Tracks*** are often used instead of wheels (e.g., tanks).

# Following Trajectories

- In locomotion we can be concerned with:
  - getting to a particular location
  - following a particular trajectory (path)
- Following an arbitrary given trajectory is harder, and it is impossible for some robots (depending on their DOF).
- For others, it is possible, but with discontinuous velocity (stop, turn, and then go again).
- A large area of traditional robotics is concerned with following arbitrary trajectories.
- **Why?**
  - Because **planning** can be used to compute optimal (and thus arbitrary) trajectories for a robot to follow to get to a particular goal location.

# Following Trajectories

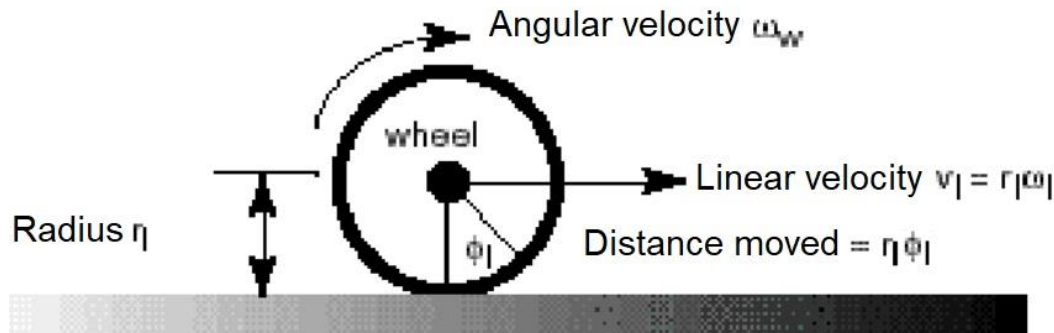
- Practical robots may not be so concerned with specific trajectories as with just getting to the goal location.
- But **trajectory planning** is a **computationally complex** process.
- **All possible** trajectories must be found (by using search) and evaluated.
- What complicates matters is that robots are not points, their geometry (i.e., turning radius) and steering mechanism (*holonomicity properties*) must be taken into account.
- This is also called **motion planning**.

# Motion Control (wheeled robots)

- Requirements
  - Kinematic/dynamic model of the robot
  - Model of the interaction between the wheel and the ground
  - Definition of required motion →
    - speed control,
    - position control
  - Control law that satisfies the requirements

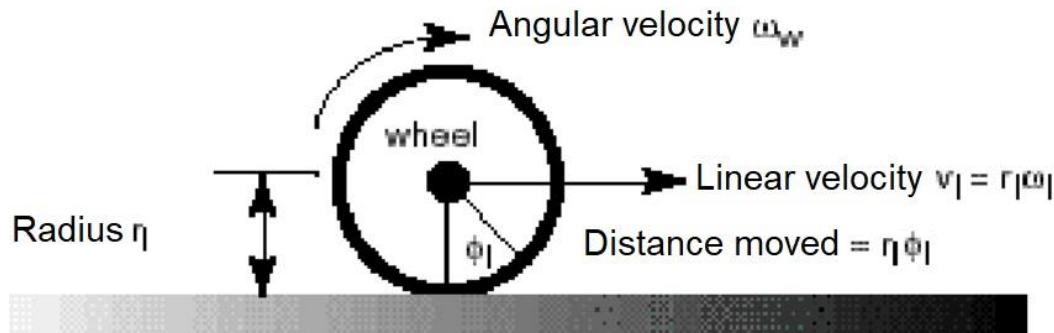
# Wheel Kinematics

- A wheel moves by rolling along the ground.
- A solid wheel touches the ground at an instantaneous contact point.
- As the wheel turns the instantaneous contact point moves both on the wheel and on the ground.
- For perfect rolling, the velocity of the contact point equals the peripheral velocity of the wheel.
- When slippage occurs, the velocity of the contact point on the ground is different to the velocity of the contact point on the wheel.



# Wheel Kinematics

- Angular displacement  $\Theta = (\Theta_s - \Theta_e)$  measured in radians (not degrees)
- Distance travelled  $d = r_w \Theta$ , where  $r_w$  is the wheel radius
- Wheel moves by  $c = 2\pi r_w$  mm per revolution.
- Linear distance moved  $d = cn + r_w \Theta$ , where  $n$  is the number of revolutions
- Linear velocity is  $v = d/t$  where  $t$  is the time period for traveling distance  $d$ 
  - Thus  $d = vt$
- Equivalent to angular linear velocity  $r_w \omega_w - f_{(slip)}$
- Linear acceleration is  $a = \Delta v / \Delta t$





# Wheel Kinematics

The linear and angular velocities of the robot are controlled by controlling the angular velocities of the wheels. The result is three cases of motion: pure translation, pure rotation and combined translation and rotation.

- **Pure translation case**

$$\omega_l = \omega_r \text{ SO } v_R = r_w \omega_l = r_w (\omega_l + \omega_r)/2 \text{ and } \omega_R = 0$$

- **Pure rotation case**

$$\omega_l = -\omega_r \text{ and } v_l = -v_r \text{ SO } v_R = 0 \text{ and } \omega_R = (v_l - v_r)/r_R = r_w(\omega_l - \omega_r)/r_R = 2\omega_l r_w/r_R$$

- **Combined translation and rotation case**

$$\omega_l > \omega_r \text{ SO } v_R = r_{\text{rot}}\omega_R$$

$$\text{and } \omega_R = v_l/(r_R + r_{\text{rot}}) - v_r/(r_R - r_{\text{rot}}) \text{ for a curve to the right}$$

where:

- $v_R$  = linear velocity of robot,
- $\omega_l$  = angular velocity of left wheel
- $\omega_r$  = angular velocity of right wheel,  $r_w$  = wheel radius, and
- $\omega_R$  = angular velocity of robot relative to centre of rotation
- $r_R$  is the radius of the robot, and  $r_{\text{rot}}$  is the radius of the arc the robot is following.

# Wheel Kinematics

- Wheel rotation is measured with rotary encoders.
- When a wheel rotates, the number of pulses is proportional to the angle rotated by the wheel.
- The linear distance *dl travelled by the periphery of the wheel is:*

$$d_l = \frac{r_l e_l}{p_l} \pm \text{slip}_l = r_l \phi_l \pm \text{slip}_l$$

where:

$r_l$  = the radius of the left wheel

$e_l$  = the angle of rotation of the left wheel

$p_l$  = left encoder pulses per degree

$\phi_l$  = angular velocity of the left wheel

$v_l$  = linear velocity of the left wheel contact point

# Wheel Kinematics

- The translation ( $D_R$ ) and rotation ( $\theta_R$ ) of the robot are calculated from the linear distances moved by the peripheries of the drive wheels.

$$\delta D_R = \frac{d_r + d_l}{2} = v_R * \delta t = D_{t+1} - D_t$$

where:

$r, l$  is the subscript for the right, left wheel.

$v$  is the linear velocity of the robot

$\delta t$  is the time period

$$\delta \theta_R = \frac{d_r - d_l}{W} = \omega_R * \delta t = \theta_{t+1} - \theta_t$$

where:

$W = 2 * r_R$  is the wheel base (the distance between the wheels).

# Wheel Kinematics

- When the wheel velocities are equal and opposite, the robot spins around its centre.
- Four wheel, car like robots, can not spin around an axis.
- However, the motion of the robot around an arc can be decomposed into rotation and translation components which are modelled by the previous equations.
- To model the kinematics of a mobile robot, we represent its location  $(x, y, \theta)$  relative to a world co-ordinate frame.
- A robot frame is located midway between the two odometry wheels.
- Sensors and wheels are also represented by co-ordinate frames relative to the robot frame.
- When the robot is turning a rotation frame is located at the centre of rotation. . .

# Wheel Kinematics

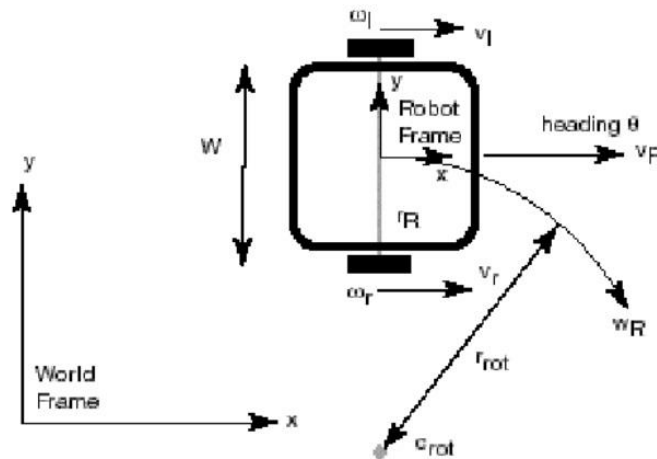
- Co-ordinate frames relative to the robot frame:

$V_R$  = linear velocity of robot

$W_R$  = angular velocity relative to centre of rotation

$c_{rot}$  = radius of the robot

$r_{rot}$  = arc radius the robot is following around  $c_{rot}$



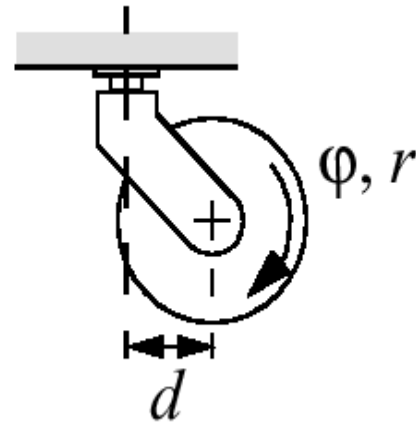
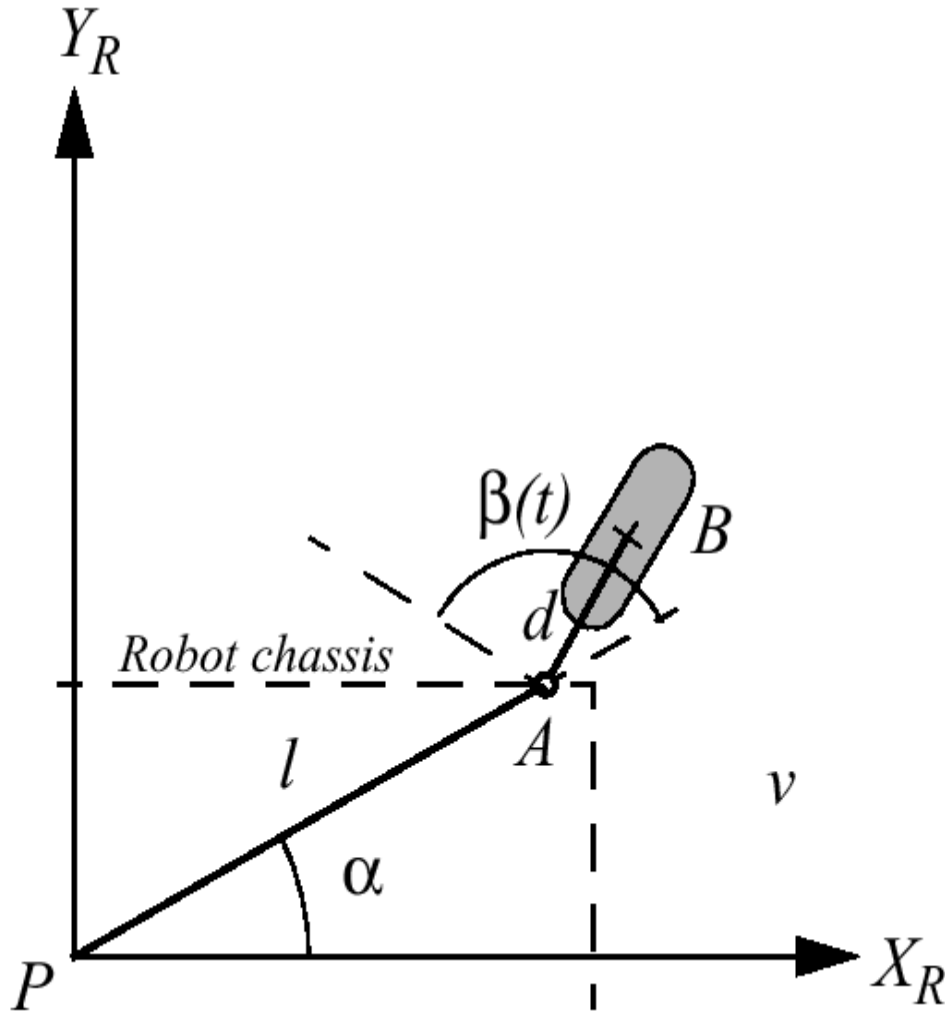
# Mobile Robot Maneuverability

- There are many design alternatives for wheeled mobile robots.
- Design problems of a single-body mobile robot include the selection of wheel types, the placement of wheels, and the determination of the kinematic parameters.
- Design objectives should be specified according to the target environments and tasks, as well as the initial and operational costs of a robot.
- Robot structures are classified according to the number of wheels and the features of commonly adopted designs.

## Wheel Kinematic Constraints:

# Castor Wheel

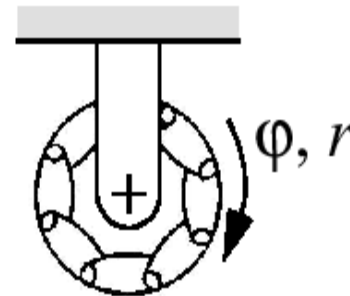
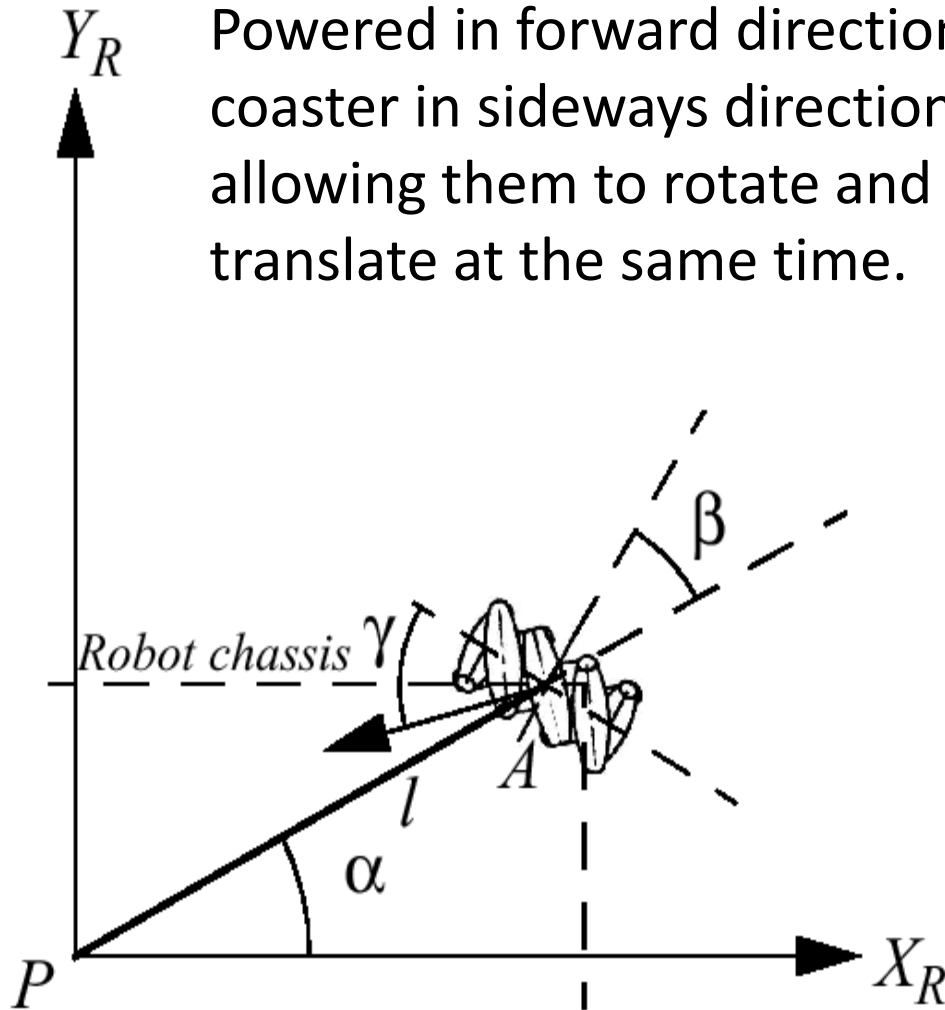
Unpowered coaster wheel which points in the direction of movement.



## Wheel Kinematic Constraints:

# Swedish Wheel

Powered in forward direction,  
coaster in sideways directions  
allowing them to rotate and  
translate at the same time.



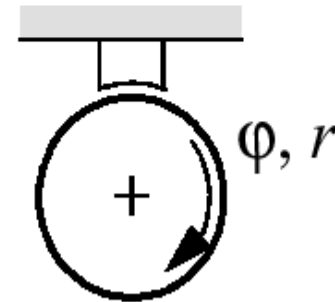
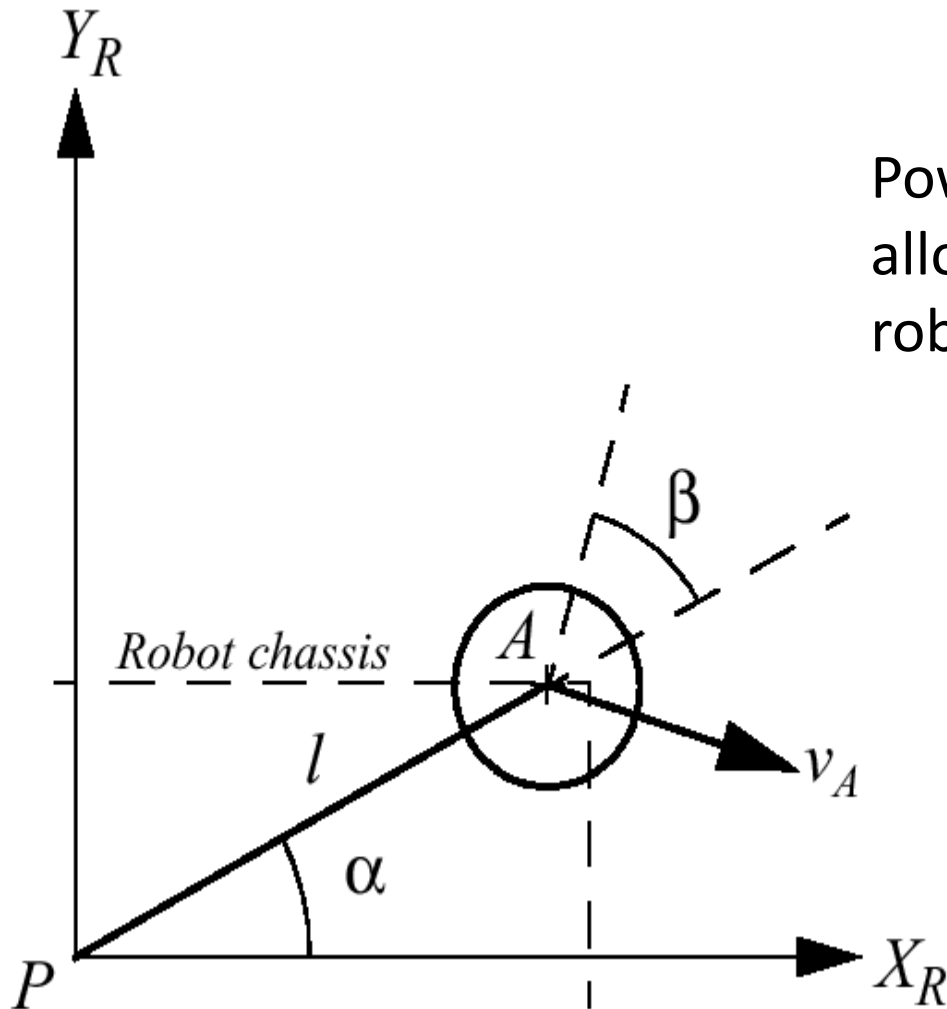
See: <https://www.youtube.com/watch?v=hLfETtUmzHg>



# Wheel Kinematic Constraints:

## Spherical Wheel

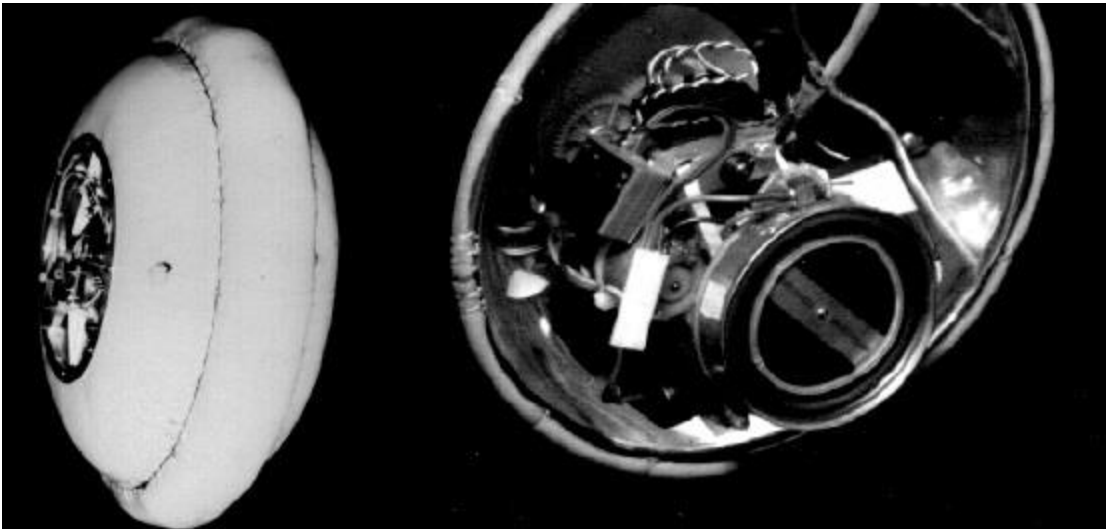
Powered in x and y directions, allowing them to drive the robot in any direction.



See: <https://www.youtube.com/watch?v=AIVUyIH-6iM>

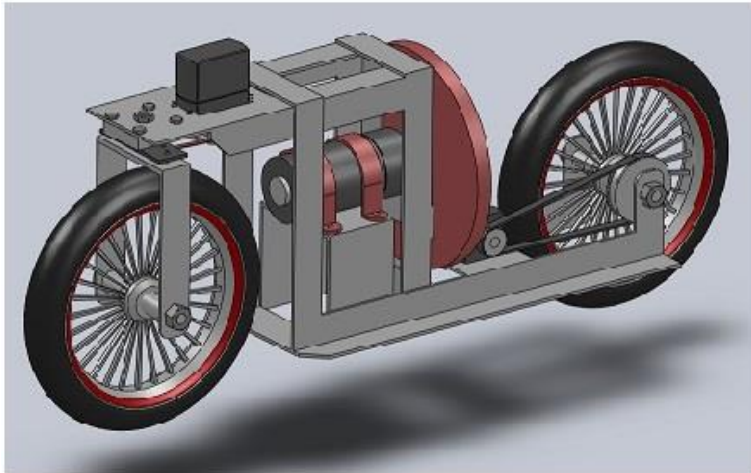
# One Wheel Robots

- A robot with a single wheel is basically unstable without dynamic control to maintain its balance. An example is a unicycle. A variation of a unicycle is a rugby-ball-shaped powered wheel.
- Single-wheel robots are rarely used in practical applications because additional balancing mechanisms are required, control is difficult, and pose estimation by pure dead reckoning is not available.



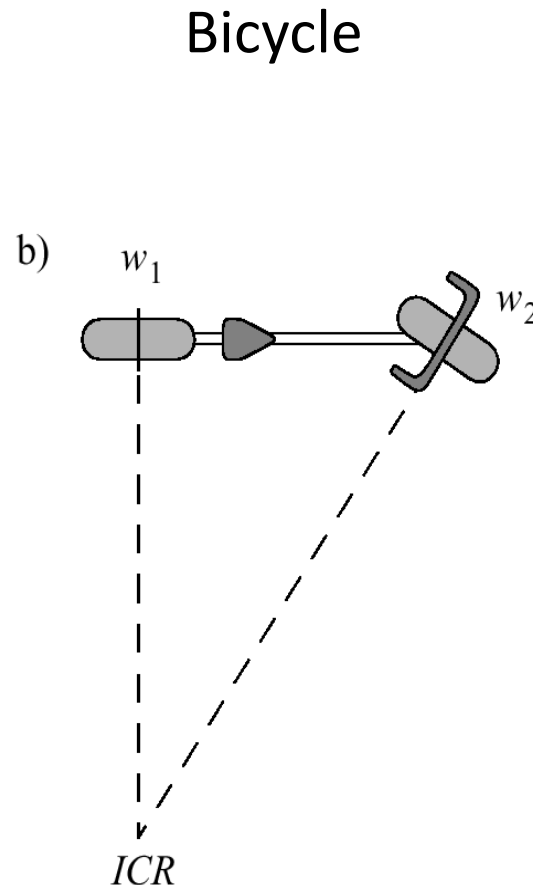
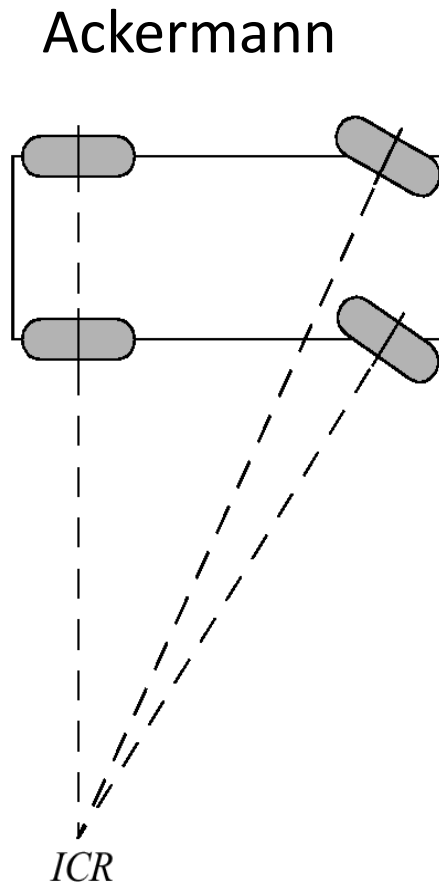
# Two Wheeled Robots

- In general, there are two types of two-wheel robots,
  - bicycle-type robot with steerable front wheel and driven a rear wheel.
  - Segway style robots with two side-by-side wheels
- The dynamic stability of a bicycle-type robot increases with its speed.
- However, a bicycle type robot is rarely used because it cannot maintain its pose when the robot stands still.
- It is possible to achieve static stability by accurately placing the centre of gravity on the wheel axle. However, it is common to apply dynamic balancing control, which is similar to the conventional control problem for an inverted pendulum.



# Mobile Robot Maneuverability

- Both Ackermann Steering and Bicycle configuration have the same Instantaneous Centre of Rotation (ICR).



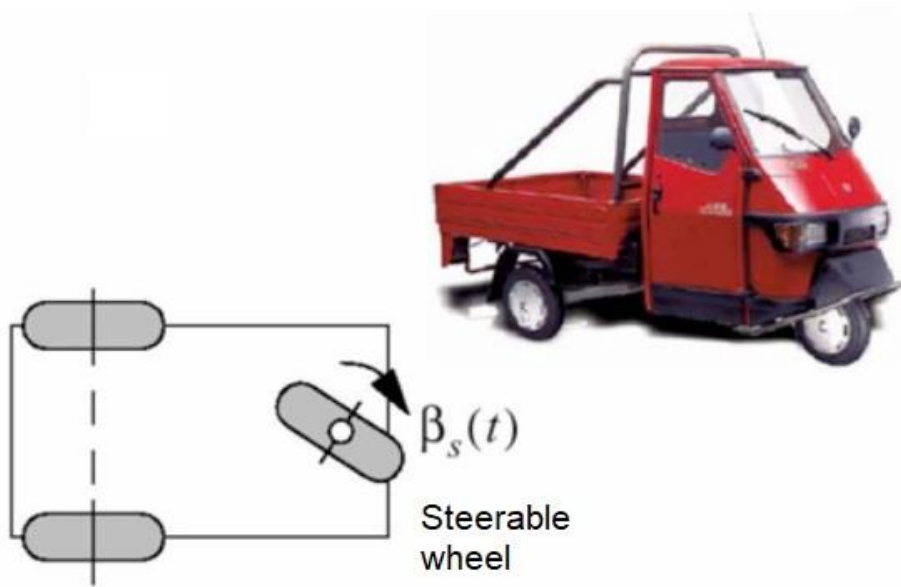
# Mobile Robot Maneuverability

- The manoeuvrability of a mobile robot is the combination:
  - of the mobility available based on the sliding constraints
  - plus additional freedom contributed by the steering
- Three wheels is sufficient for static stability
  - additional wheels need to be synchronised
  - this is also the case for some arrangements with three wheels
- To rank a mobile robot we use the degree of maneuverability:

➤ <i>Degree of mobility</i>	$\delta_m$
➤ <i>Degree of steerability</i>	$\delta_s$
➤ <i>Robots maneuverability</i>	$\delta_M = \delta_m + \delta_s$

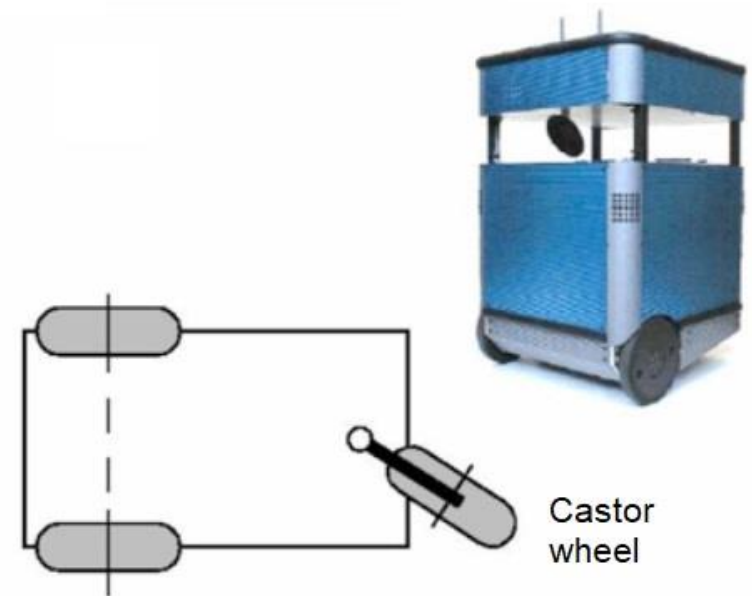
# Mobile Robot Maneuverability

- Degree of Manoeuvrability:  $\delta M = \delta m + \delta s$
- Two robots with same  $\delta M$  are *not necessary equal*
- Example: Differential drive and Tricycle



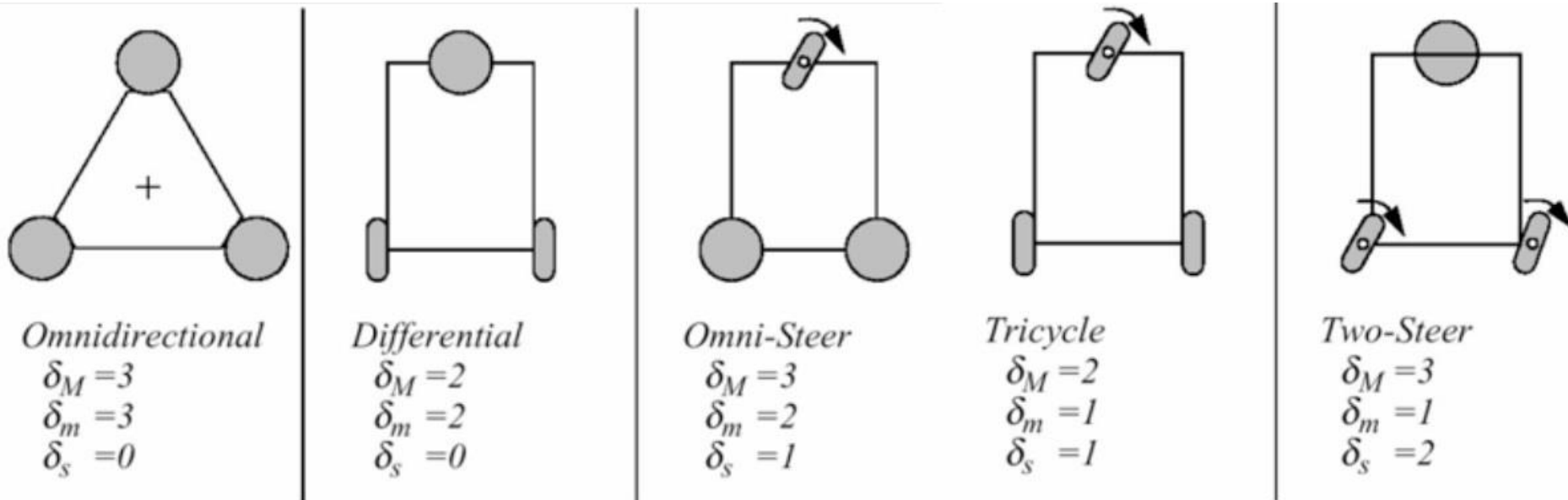
Tricycle Example

## Differential Drive



# Degree of Manoeuvrability 1

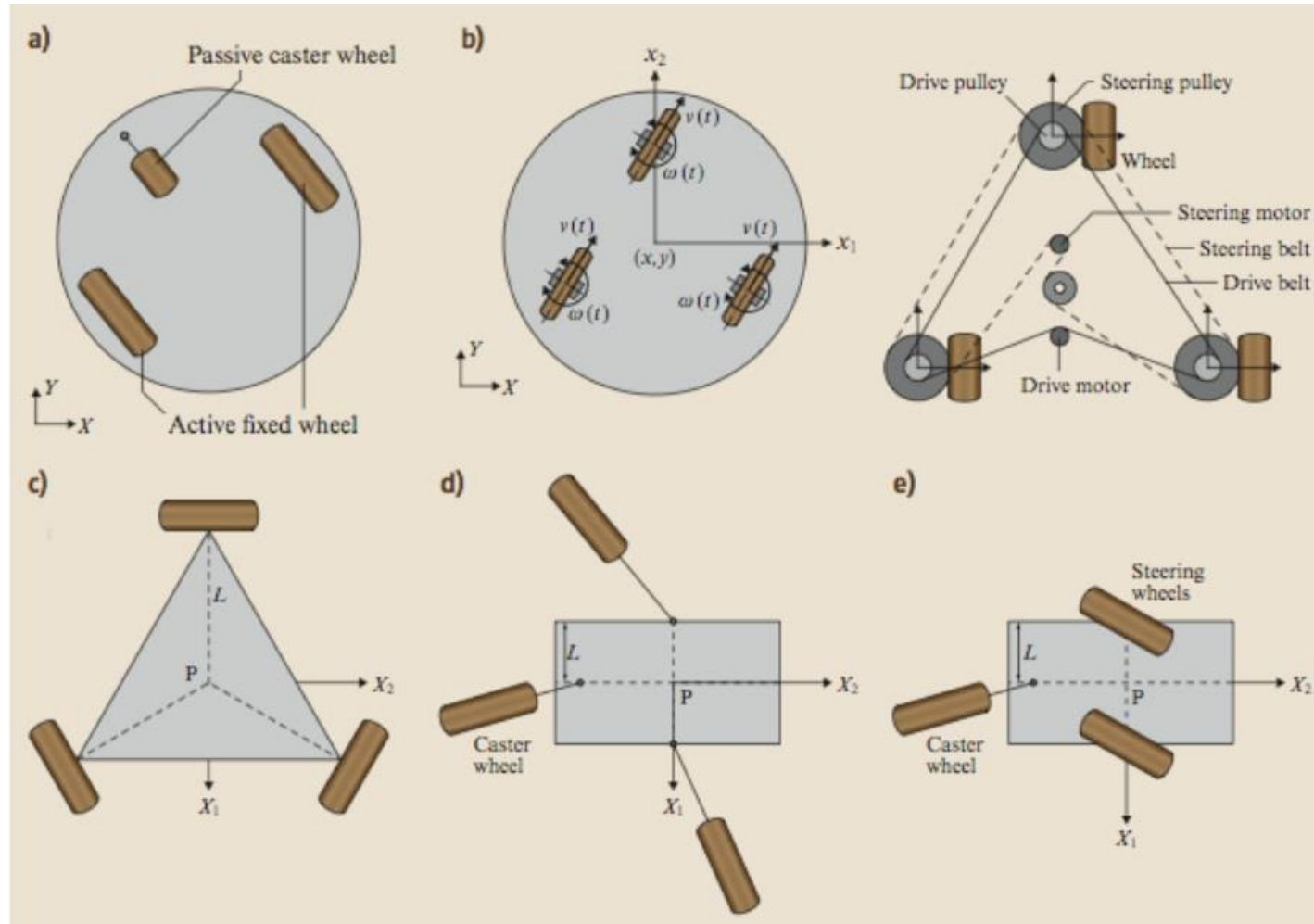
- For any robot with  $\delta M = 2$  the ICR is always constrained to lie on a line with radius  $r$ .
- For any robot with  $\delta M = 3$  the ICR is not constrained and can be set to any point on the plane.



Five Basic Types of Three-Wheel Configurations

# Degree of Manoeuvrability 2

- 3 wheel robot examples:

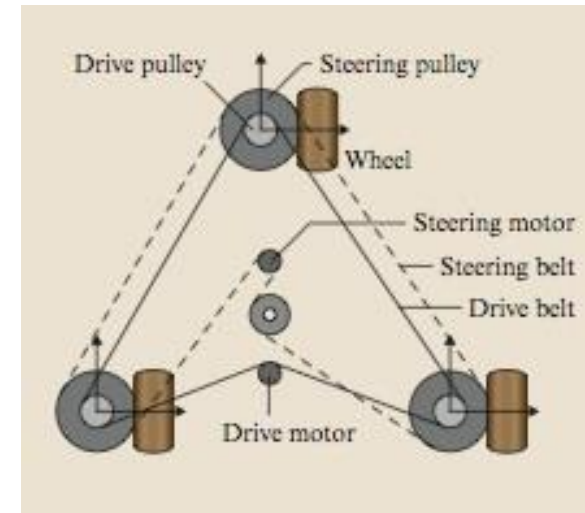


- (a) Two-wheel differential drive, (b) synchronous drive, (c) omnimobile robot with Swedish wheels, (d) omnimobile robot with active caster wheels, (e) omnidirectional robot with active steerable wheels



# Synchronous-Drive Robot

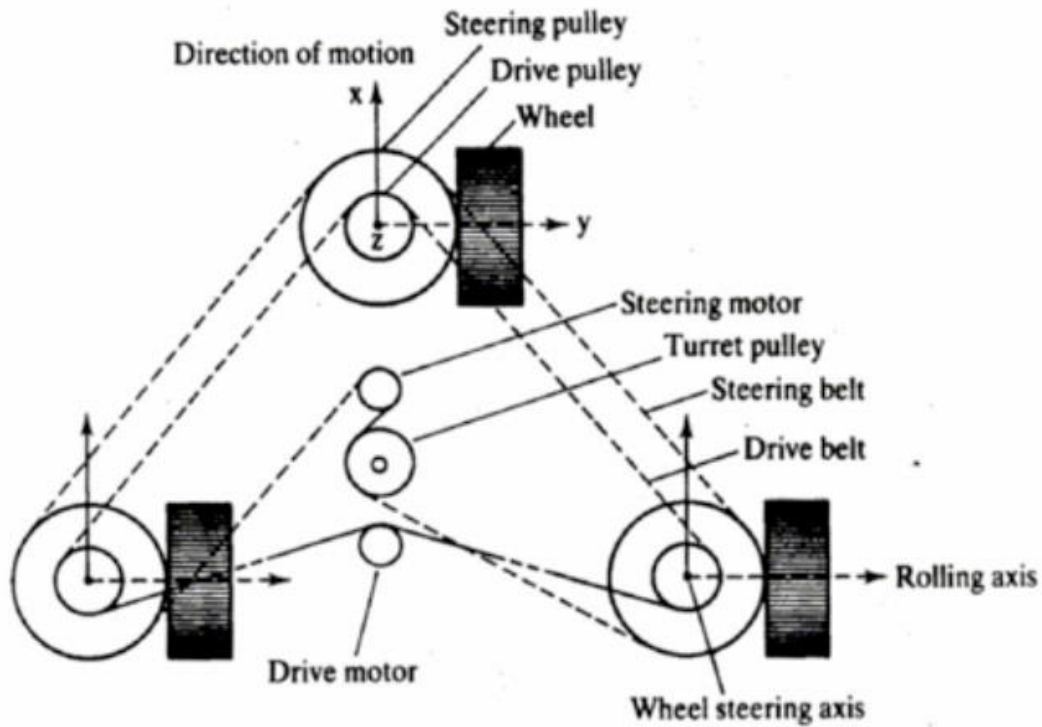
- A synchronous-drive robot can be built by using centred or off-centred orientable wheels.
- The steering and driving motions of each wheel are mechanically coupled by chains or belts, and the motions are actuated synchronously, so the wheel orientations are always identical.
- Therefore, omnidirectional motion, i.e., motion in any direction, can be achieved by steering the wheel orientations to the desired velocity direction. However, the orientation of the robot chassis cannot be changed.
- The most significant advantage of the synchronous-drive robot is that omnidirectional movement can be achieved by using only two actuators.



# Synchronous-Drive Robot

- A synchronous-drive robot example:

$$\delta_M = \delta_m + \delta_s = 1 + 1 = 2$$



# Degrees of Freedom (DOF)

- Manoeuvrability  $\delta M$  is equivalent to the vehicle's degree of freedom (*DOF*). *This is often* equivalent to the number of servos on the robot.
- This should not be confused with the degree of a vehicle's freedom in its environment.
- This is often referred to as the differentiable degrees of freedom (*DDOF*) *and is equivalent to* the robot's independently achievable velocities.
- Example:
  - Bicycle:
    - $\delta M = \delta m + \delta s = 1 + 1$   $DDOF = 1$ ;  $DOF = 2$
  - Omni Drive:
    - $\delta M = \delta m + \delta s = 3 + 0$   $DDOF = 3$ ;  $DOF = 3$

# Omni Drive Robots

:



Example: <https://www.youtube.com/watch?v=1BEXBWAFYew>

# Mobile Robot Workspace: Degrees of Freedom, Holonomy

- DOF *degrees of freedom*:
  - Robots ability to achieve various poses
- DDOF *differentiable degrees of freedom*:
  - Robots ability to achieve various path

$$DDOF \leq \delta_m \leq DOF$$

- Holonomic Robots
  - A holonomic kinematic constraint can be expressed as an explicit function of position variables only.
  - A non-holonomic constraint requires a different relationship, such as the derivative of a position variable.
  - Fixed and steered standard wheels impose non-holonomic constraints.

# Beyond Basic Kinematics

- Speed, force & mass → dynamics
- Important when:
  - High speed
  - High/varying mass
  - Limited torque
- Friction, slip, skid, ..
- How to actuate: “motorization”
- How to get to some goal pose: “controllability”

# Motion Control (kinematic control)

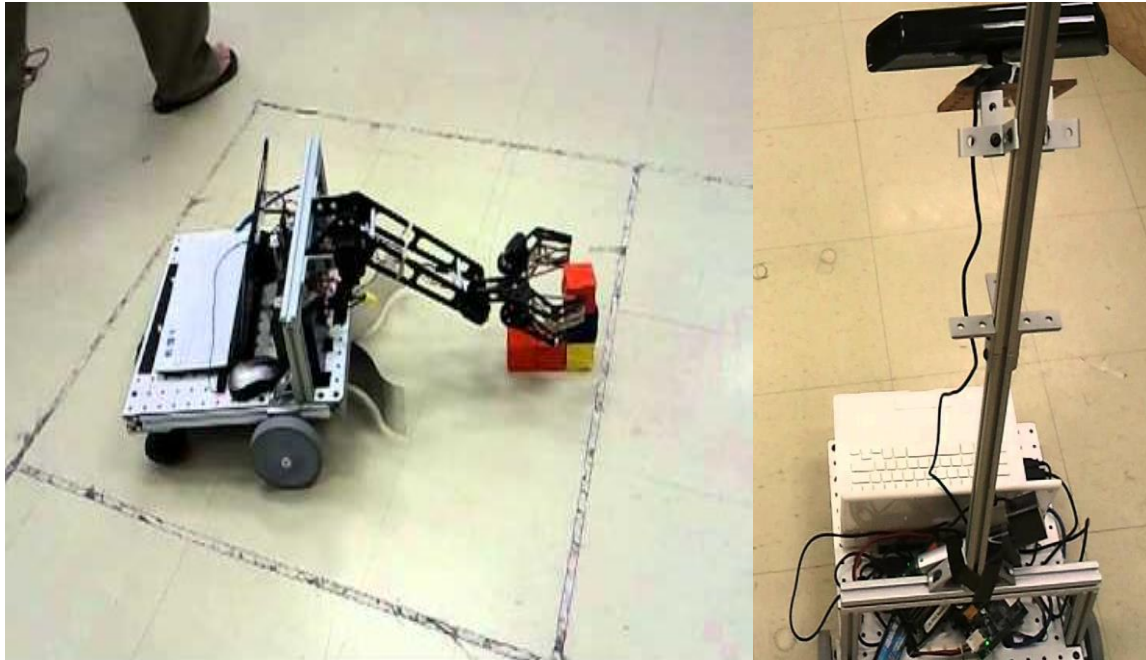
- Objective:
  - follow a trajectory
  - position and/or velocity profiles as function of time.
- Motion control is not straightforward:
  - mobile robots are non-holonomic systems!
- Most controllers are not considering the dynamics of the system
  - need robot operating systems software development platforms for dynamic systems.

# Challenges with programming robots

1. The world is asynchronous
2. Robots must manage significant complexity
3. Robot hardware requires abstraction



**NEED AN ARCHITECTURE WHICH CAN SUPPORT MORE COMPLEX HARDWARE AND SOFTWARE COMPONENTS.**



Goal: Develop “big” software for robots

Need an architecture which can support more complex hardware and software components.



MIT Urban Challenge Vehicle

# Problem 1: Sequential Programming

```
goForward(1);  
turnLeft(Math.PI/2);  
Image image = camera.getImage();  
double distance = computeDistanceToObject(image);  
goForward(distance - 1);  
(x, y) = getMyPositionFromTheEncoderCounts();  
...
```

What happens if an obstacle appears while you are going forward?

What happens to the encoder data while you are turning?

What if some other module wants the same data?

# Solution 1: “Callbacks”

## *Callback:*

Function which is called whenever data is available for processing.

An *asynchronous callback* can happen at any time.

## *Examples:*

Run the relevant *callback* function whenever:

- An image is read from the camera
- The odometry sensor reports new data

```
void imageCallback(ImageMessage image)
    // process the latest image

void odometryCallback(OdometryMessage data)
    // handle latest odometry data

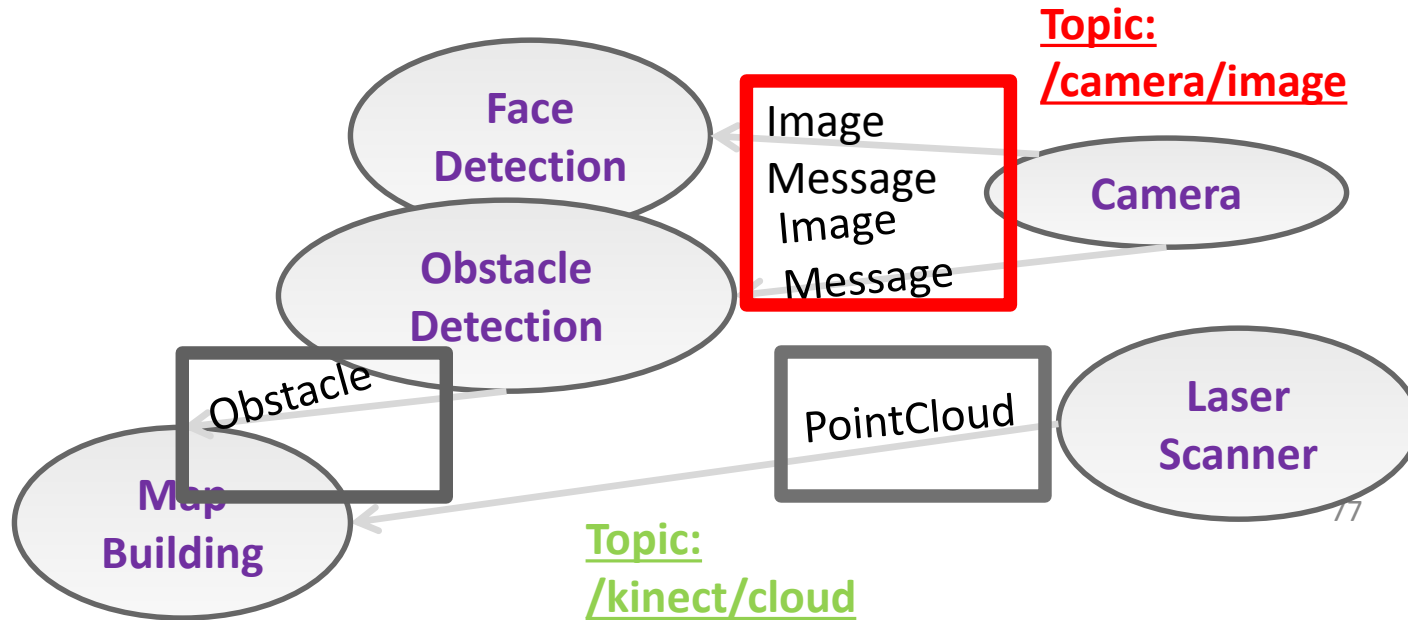
void main()
    initialize();
    subscribe("image_msgs", imageCallback);
    subscribe("odometry_msgs",
odometryCallback);
```

## Solution 2: Organizing code

*Separate processes:* Cameras, Odometry, Laser Scanner, Map Building can all be separated out: they'll interact through an interface

*Interfaces:* Software processes (“nodes” in ROS) communicate about shared “topics” in ROS

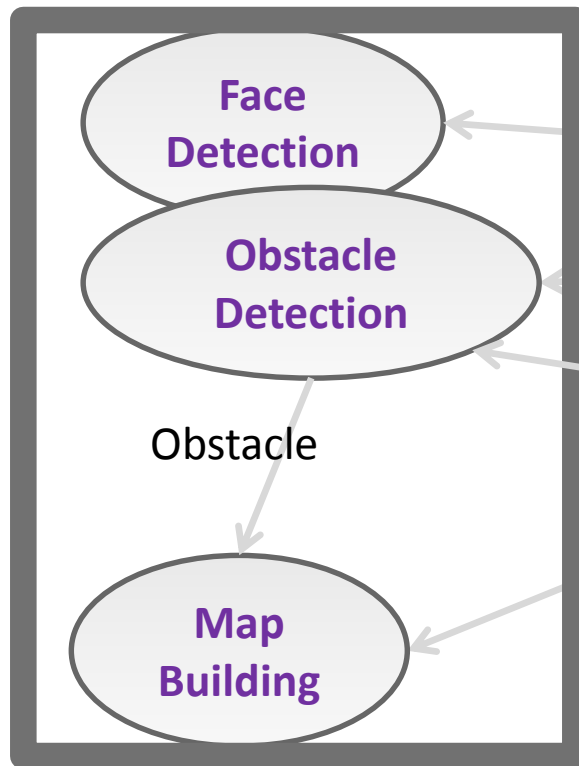
*Publish/Subscribe*: Have each module receive *only* the data (messages) it requests



# Problem 3: Hardware dependent code

## Solution 3: Abstracting hardware

Hardware-Independent  
Software



Device-Specific  
Drivers

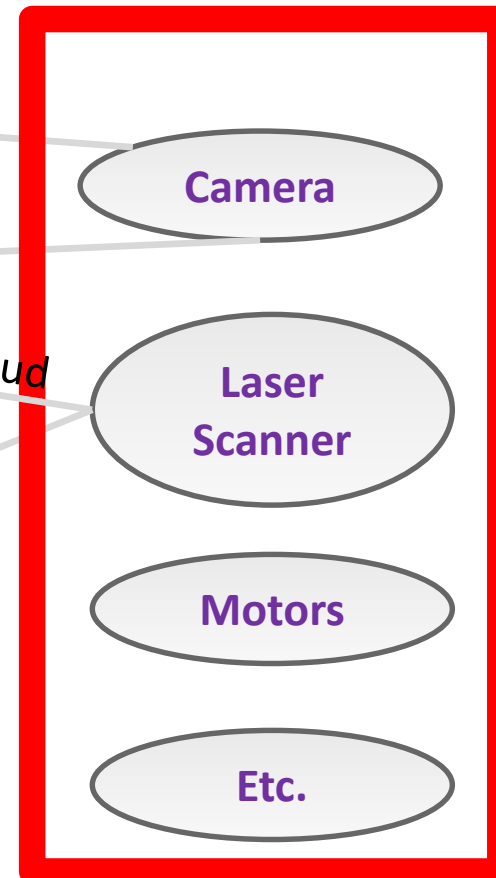

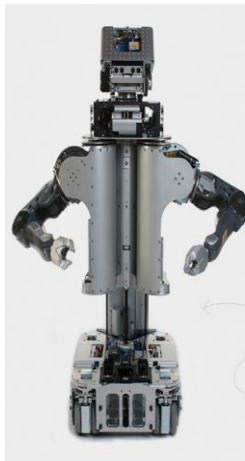


Image  
Message  
Image  
Message  
PointCloud  
PointCloud

  
**Interface**

# Result: Reusable code!



PR2



Roomba



Care-O-bot 3

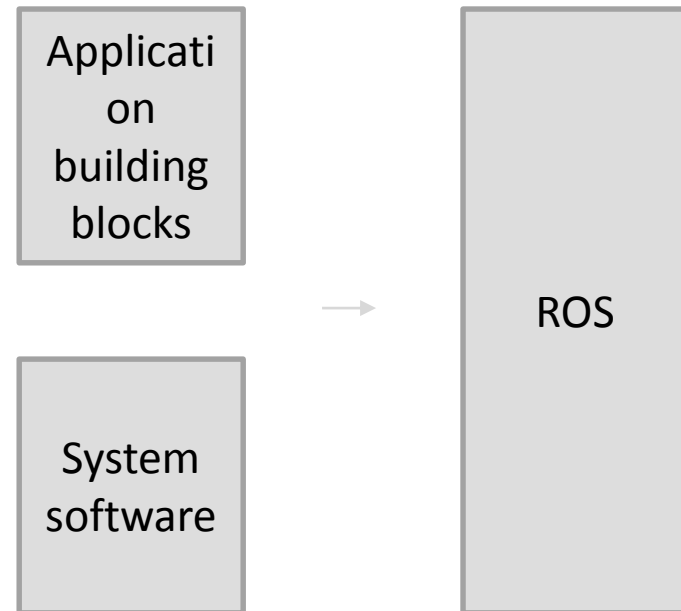
# Robot Operating System (ROS)

- What is ROS?
  - A “Meta” Operating System.
    - Open source
    - Runs in Linux (esp. Ubuntu)
    - Ongoing Windows implementation
  - Agent based (nodes)
  - Message passing
    - Publish
    - Subscribe
    - Services via remote invocation
  - Supports numerous programming languages (C++, Python, Lisp, Java)



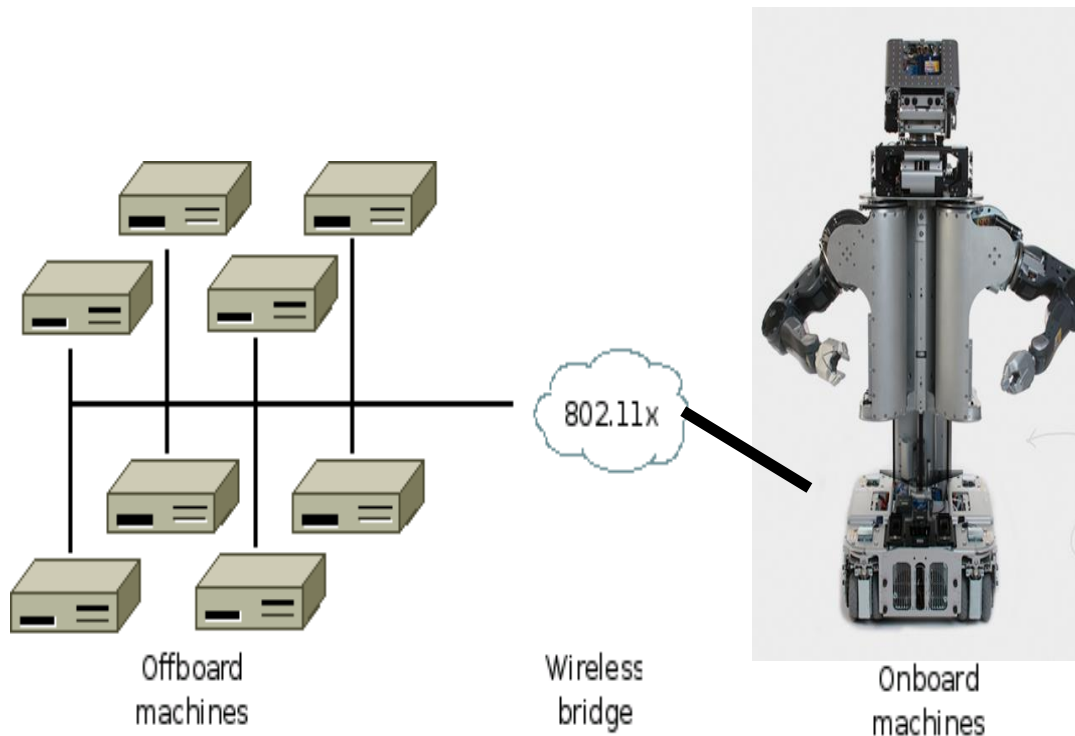
# What is ROS?

- Low level device abstraction
  - Joystick
  - GPS
  - Camera
  - Controllers
  - Laser Scanners
  - ...
- Application building blocks
  - Coordinate system transforms
  - Visualization tools
  - Debugging tools
  - Robust navigation stack (SLAM with loop closure)
  - Arm path planning
  - Object recognition
  - ...



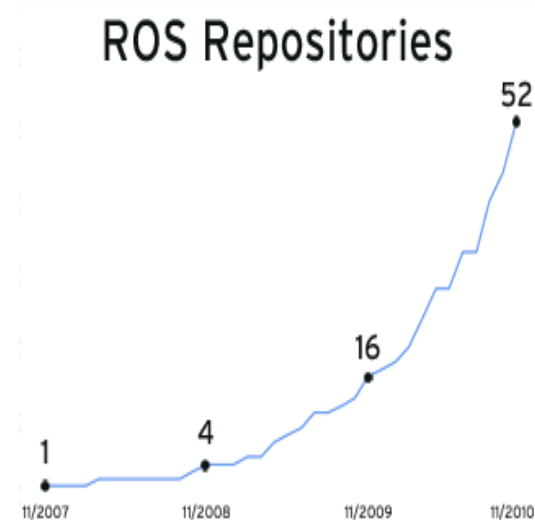
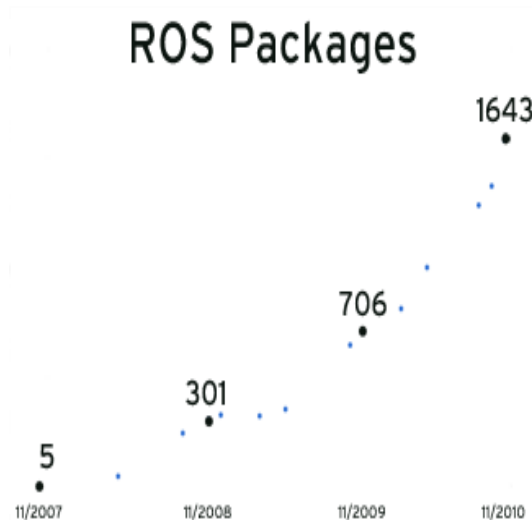
# What is ROS?

- Software management (compiling, packaging)
- Remote communication and control



# What is ROS?

- Founded by Willow Garage
- Exponential adoption
- Countless commercial, hobby, and academic robots use ROS (<http://wiki.ros.org/Robots>)



# Tools-based software design

Tools for:

- Building ROS nodes
- Running ROS nodes
- Viewing network topology
- Monitoring network traffic

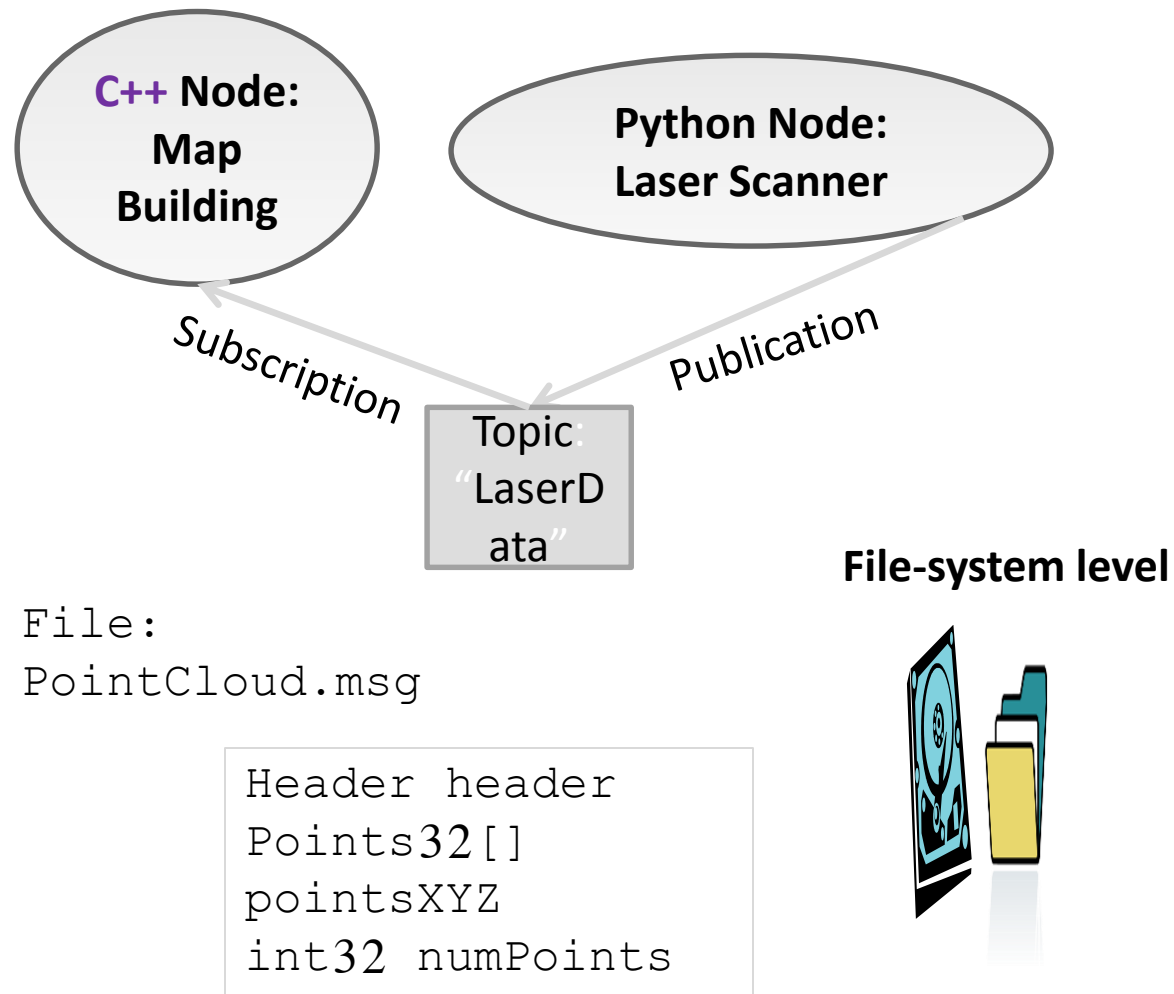
Many cooperating processes, instead of a single monolithic program.

# ROS Philosophical goals

- “Hardware agnosticism”
- Peer to peer
- Tools based software design
- Multiple language support (C++/Java/Python)
- Lightweight: runs only at the edge of your modules
- Free
- Open source
- Suitable for large scale research and industry

# Multiple language support

- ROS is implemented natively in each language.
- Quickly define messages in language-independent format.



# Lightweight

- Encourages standalone libraries with no ROS dependencies:  
*Don't put ROS dependencies in the core of your algorithm!*
- Use ROS only at the *edges* of your interconnected software modules:  
Downstream/Upstream interface
- ROS re-uses code from a variety of projects:
  - OpenCV: Computer Vision Library
  - Point Cloud Library (PCL): 3D Data Processing
  - OpenRAVE: Motion Planning

**ROS Community**



Carnegie  
e  
Mellon

# Free & Open Source

- BSD License : Can develop commercial applications
- Drivers (Kinect and others)
- Perception, Planning, Control libraries
- MIT ROS Packages : Kinect Demos, etc
- Interfaces to other libraries: OpenCV, etc



# Summary

- Locomotion
  - Legs vs wheels
  - Static vs dynamic
- Wheeled Robots
  - Efficient
  - Many configurations and Degrees of Mobility
- Programming challenges
  - The world is schronous
  - Control and the environment is complex
- Solution
  - Callbacks
  - Organization
  - Abstraction