Representing and Modeling Space

McGill COMP 765

Jan 9th, 2019

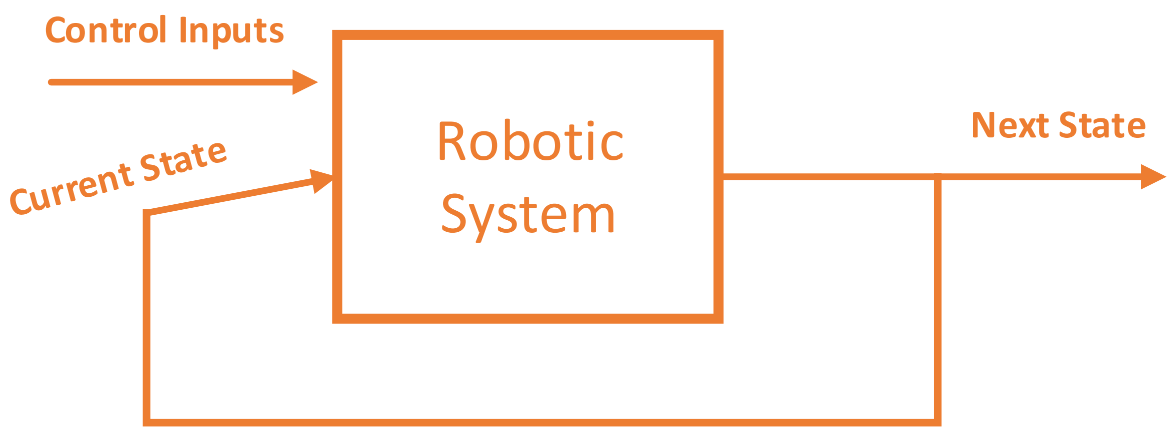
* The topology of quantum worm-holes will be left for self-study.

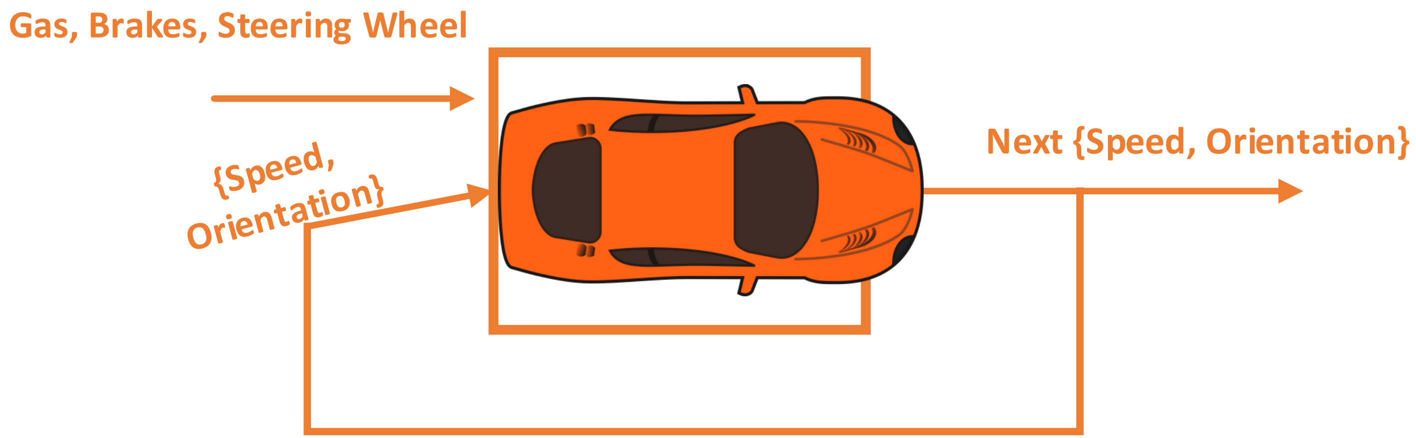
Goals today

- Brief beginning on models of a robot's movements
- Brief beginning on models of robotic sensors
- Some first models to represent the space around the robot (environment)
- Introduction of key robotics problems:

Physical models of how systems move

Kinematics & Dynamics:
Idealized models of
robotic motion



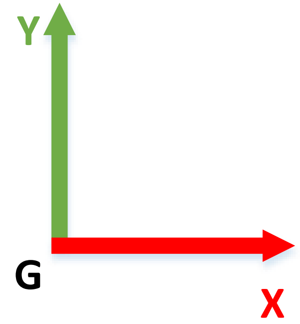


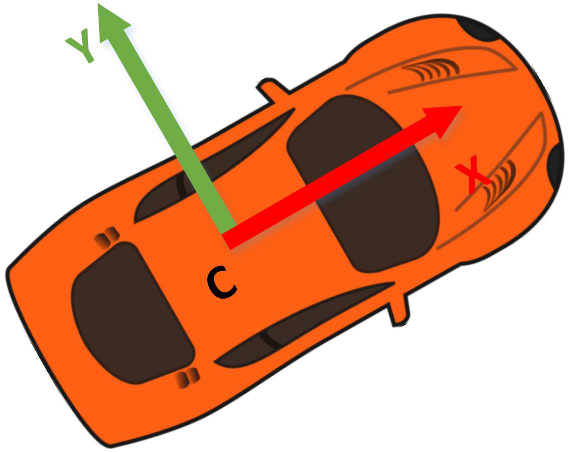
Kinematics and Dynamics

- Kinematics considers models of locomotion independently of external forces and control:
  - For example, how the speed of a cars wheels affect the motion of its chassis.

- Dynamics considers models of locomotion as functions of their control inputs and state – considering all present forces:
  - For example, how a quadrotor will move when the rotor’s fight gravity

Example: Kinematics of a simple car - state





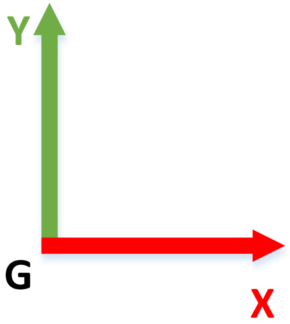
$$\mathbf{x} = [{}^{G}p_x, {}^{G}p_y, {}^{G}\theta]$$

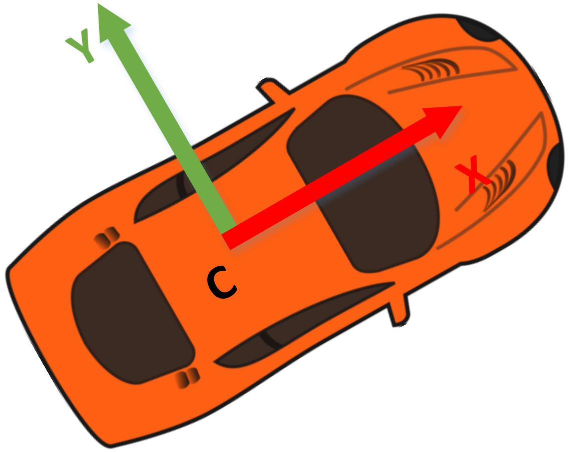
State = [Position and orientation]
Position of the car's frame of reference C with respect to a fixed frame of reference G, expressed in frame G.
The angle is the orientation of frame C with respect to G.

Note on Inertial frames of reference

- G, the global frame of reference is fixed, i.e. with zero velocity in our previous example.
- If a robot's pose is known in the global frame, life is good:
  - It can correctly report its position
  - We can compute the direction and distance to a goal point
  - We can avoid collisions with obstacles also known in global frame
- We may also know the pose in another frame: e.g. of a map, sensor, or relative to it's starting point. These are often sub-steps for us.

Example: the dynamics of a simple car - controls

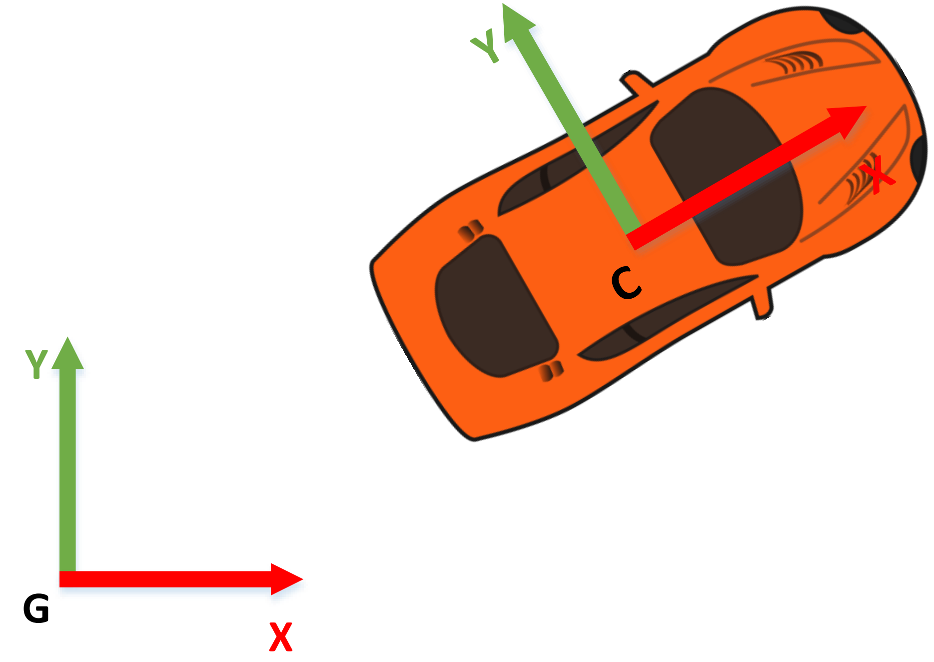




$$\mathbf{u} = [{}^{C}v_x, {}^{C}\omega_z]$$

Controls = [Forward speed and angular velocity]
Linear velocity and angular velocity of the car's frame of reference C with respect to a fixed frame of reference G, expressed in coordinates of C.

The dynamical system of a simple car



$$
\begin{aligned}
\dot{p}_x &= v_x \cos(\theta) \\
\dot{p}_y &= v_x \sin(\theta) \\
\dot{\theta} &= \omega_z
\end{aligned}
$$

Note: reference frames have been removed for readability.

Kinematics and Dynamics Interfaces: both have a forward and inverse query

- Kinematics:
  - Forward – what robot position results from a given configuration? Example: given the 6 joint angles, where is the end effector of an arm in 3-space.
  - Inverse – which configuration(s) gives a desired position? Example: to grasp a point in 3-space, solve for the right joint angles.
- Dynamics:
  - Forward – what robot motion will result from given input forces? Example: predict the motion of a quadrotor spinning only one of its propellers.
  - Inverse – which input forces will result in a desired robot motion? Example: in order to stabilize gravity, solve for the force is needed at each propeller.

Special case of simple car: Dubins car

- Can only go forward
- Constant speed

$$ {}^{C}v_x = \text{const} > 0 $$

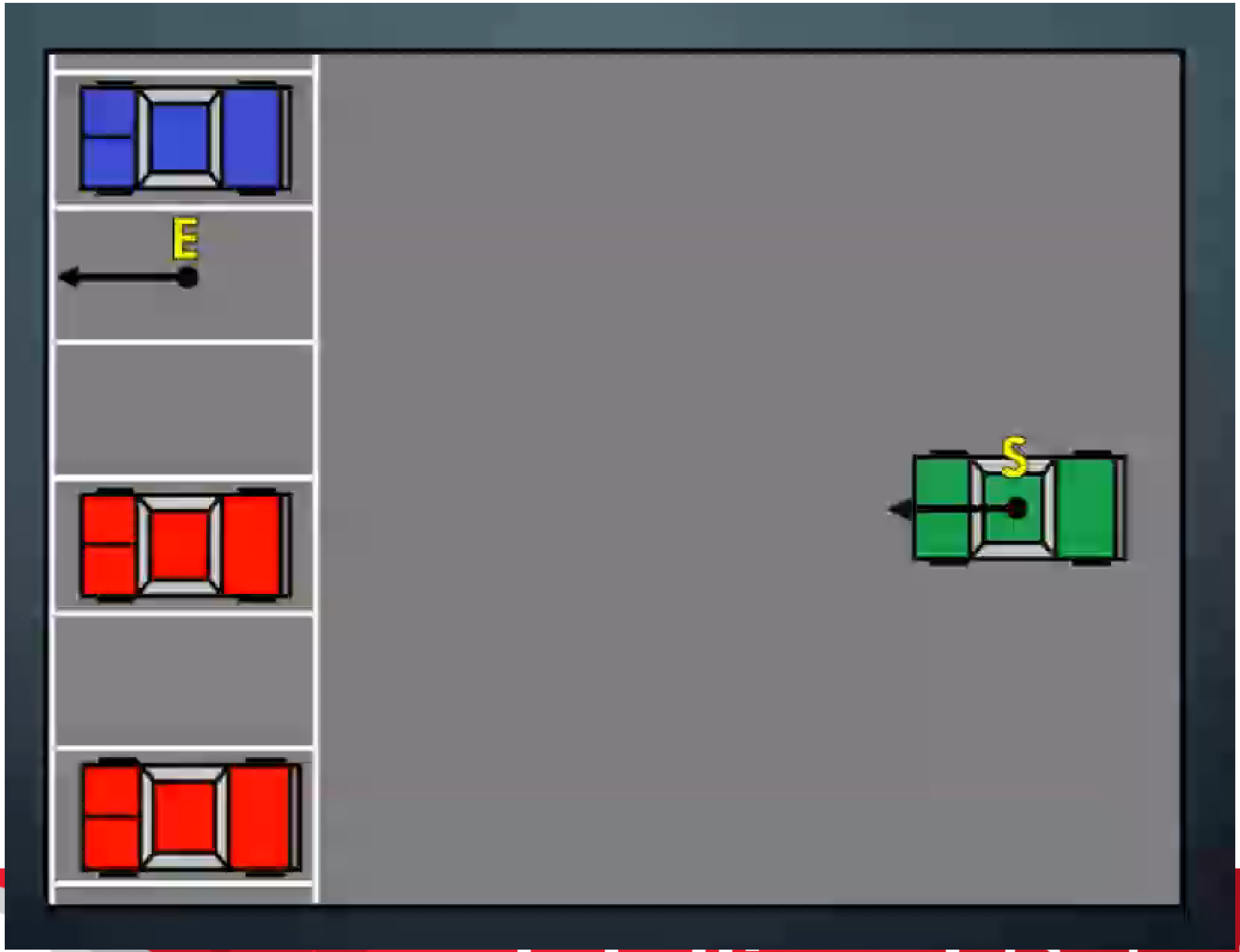
- You only control the angular velocity

Special case of simple car: Dubins car

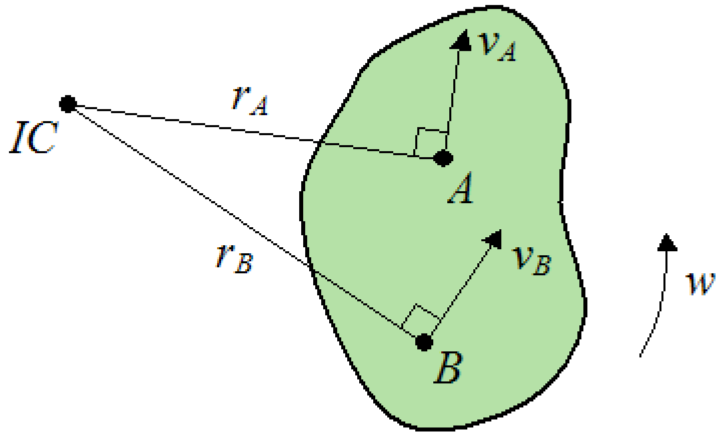
- Can only go forward
- Constant speed

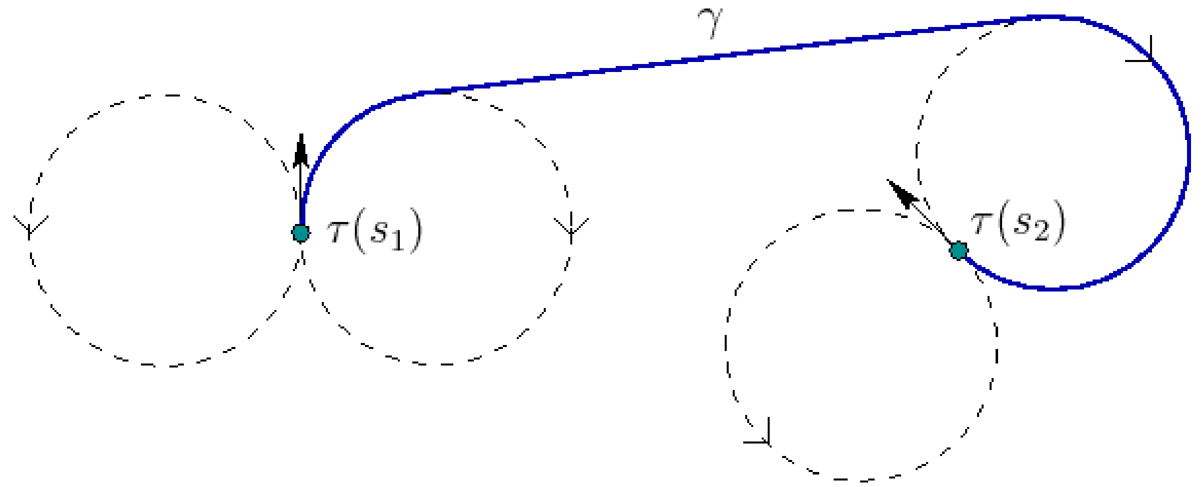
$$ {}^{C}v_x = \text{const} > 0 $$

- You only control the angular velocity



Instantaneous Center of Rotation

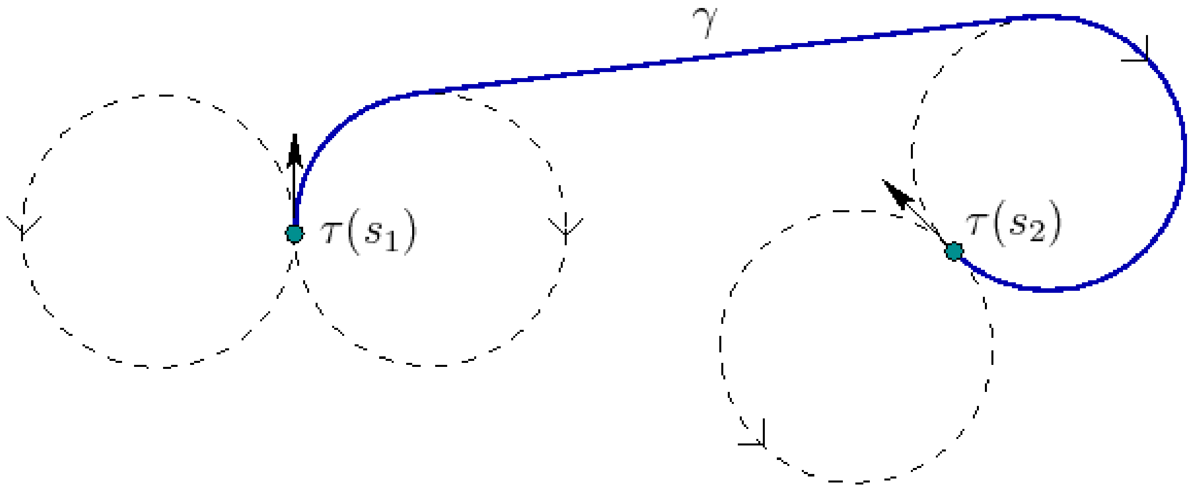




IC = Instantaneous Center of Rotation
The center of the circle circumscribed by the turning path.
Undefined for straight path segments.

Dubins car: motion primitives

- Optimal paths of the car can be decomposed to L(eft), R(ight), S(traight) segments.
- Planning paths of this nature for various start/endis already an interesting challenge worthy of some thinking (often an assignment)



RSR path

Dubins car (boat, plane, etc)

- We can also model idealized airplanes and boats in this fashion
- Elements of this reasoning exist in mission planners (ArduPilot, MavLinks, ROS software, air traffic control systems)

- This is also just one constraint out of many. What can we learn more generally about when the kinematic structure will be interesting?

Holonomic constraints

- Equality constraints on the state of the system, but not on the higher-order derivatives:

$$\mathbf{f}(\mathbf{x}, t) = 0$$

- For example, if you want to constrain the state to lie on a circle:

$$||\mathbf{x}||^2 - 1 = 0$$

- Another example: train tracks are a holonomic constraint.

Holonomic constraints

- Under only holonomic constraints, all of the local neighborhood is reachable

- We will search for time-varying shortest paths in the state space to move farther

$$\mathbf{f}(\mathbf{x}, t) = 0$$

Non-holonomic constraints

- Equality constraints that involve the derivatives of the state (e.g. velocity) in a way that it cannot be integrated out into holonomic constraints, i.e.

$$\mathbf{f}(\mathbf{x}, \dot{\mathbf{x}}, t) = 0$$

but not

$$\mathbf{f}(\mathbf{x}, t) = 0$$

Non-holonomic constraints

- With the time derivative (first or higher order) in the constraint, we can only reach a sub-set of the state-space neighborhood immediately
- Optimal paths lie on a manifold with more complex geometry which leads to more interesting estimation, planning and control
- We will primarily be interested with algorithms that can handle non-holonomic constraints

$$\mathbf{f}(\mathbf{x}, \dot{\mathbf{x}}, t) = 0$$

The Dubins car is non-holonomic

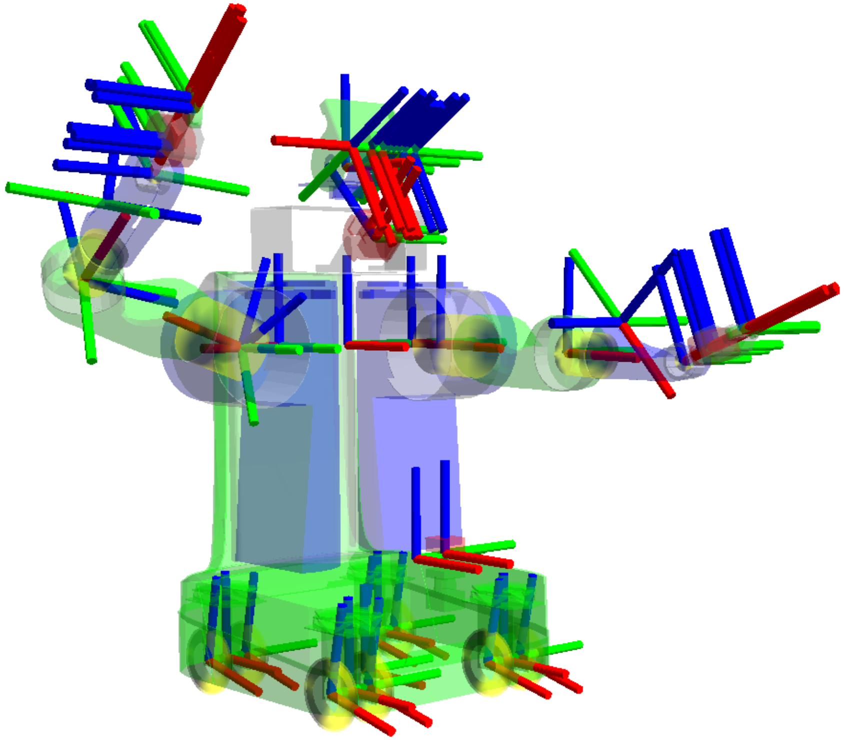
- Dubins car is constrained to move straight towards the direction it is currently heading. It cannot move sideways. It needs to "parallel park" to move laterally.

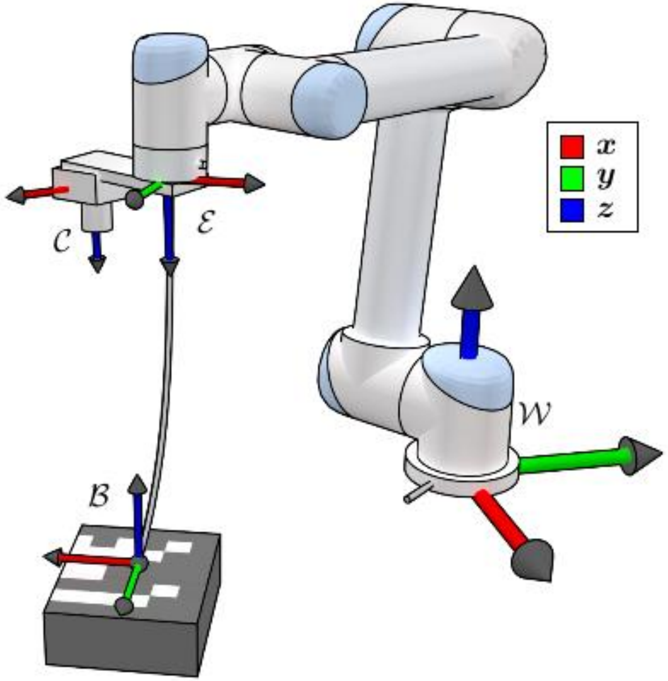
- In a small time interval dt the vehicle is going to move by δp_x and δp_y in the global frame of reference. Then from the dynamical system:

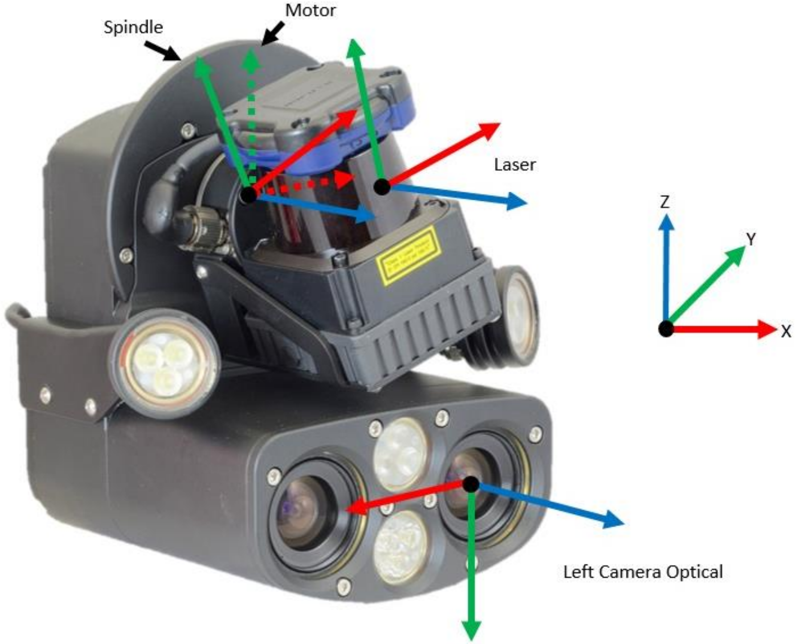
$$\begin{aligned} \delta p_x \sin(\theta) &= v_x \cos(\theta)\sin(\theta) \\ \delta p_y \cos(\theta) &= v_x \sin(\theta)\cos(\theta) \end{aligned} \Rightarrow \delta p_x \sin(\theta) = \delta p_y \cos(\theta) \Rightarrow v_x \sin(\theta) = v_y \cos(\theta)$$

Car is constrained to move along the line of current heading, i.e. non-holonomic

3D frames of reference are everywhere in robotics





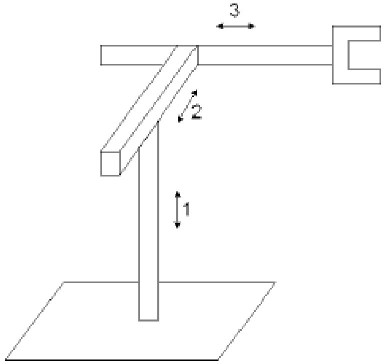


Manipulators

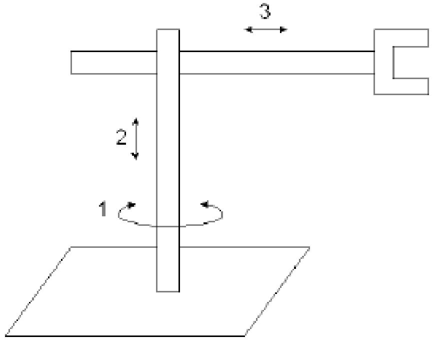
- Robot arms, industrial robot
  - Rigid bodies (links) connected by joints
  - Joints: revolute or prismatic
  - Drive: electric or hydraulic
  - End-effector (tool) mounted on a flange or plate secured to the wrist joint of robot

Manipulators

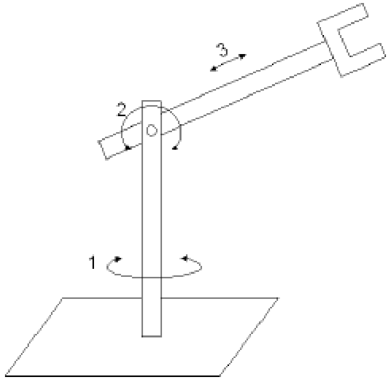
- Robot Configuration:



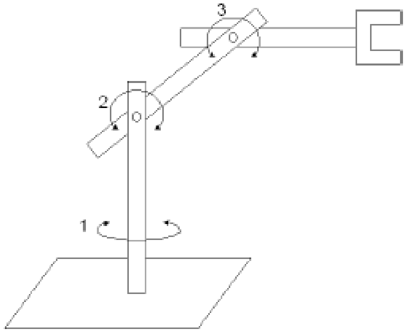
Cartesian: PPP



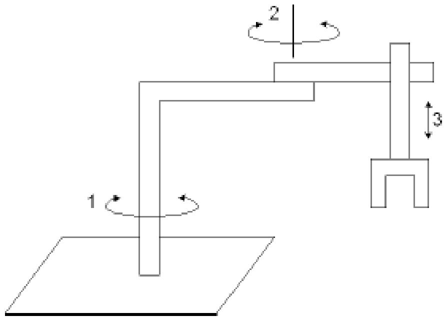
Cylindrical: RPP



Spherical: RRP

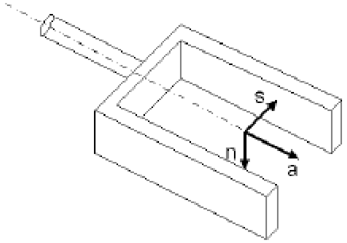


Articulated: RRR



SCARA: RRP

(Selective Compliance
Assembly Robot Arm)



Hand coordinate:

n: normal vector; **s**: sliding vector;

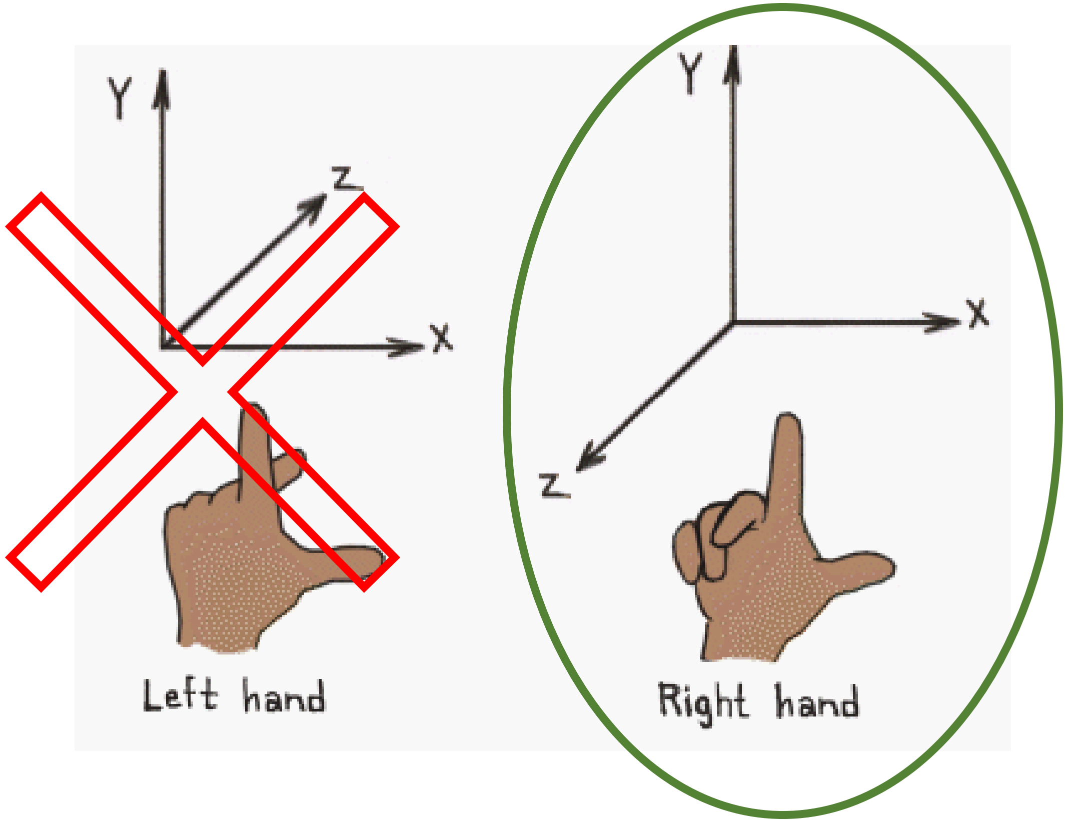
a: approach vector, normal to the tool mounting plate

Manipulators

- Robot Specifications
  - Number of Axes
    - Major axes, (1-3) => Position the wrist
    - Minor axes, (4-6) => Orient the tool
    - Redundant, (7-n) => reaching around obstacles, avoiding undesirable configuration
  - Degree of Freedom (DOF)
  - Workspace
  - Payload (load capacity)
  - Precision v.s. Repeatability

Which one is more important?

Right-handed vs left-handed frames



Unless otherwise specified, we use right-handed frames in robotics

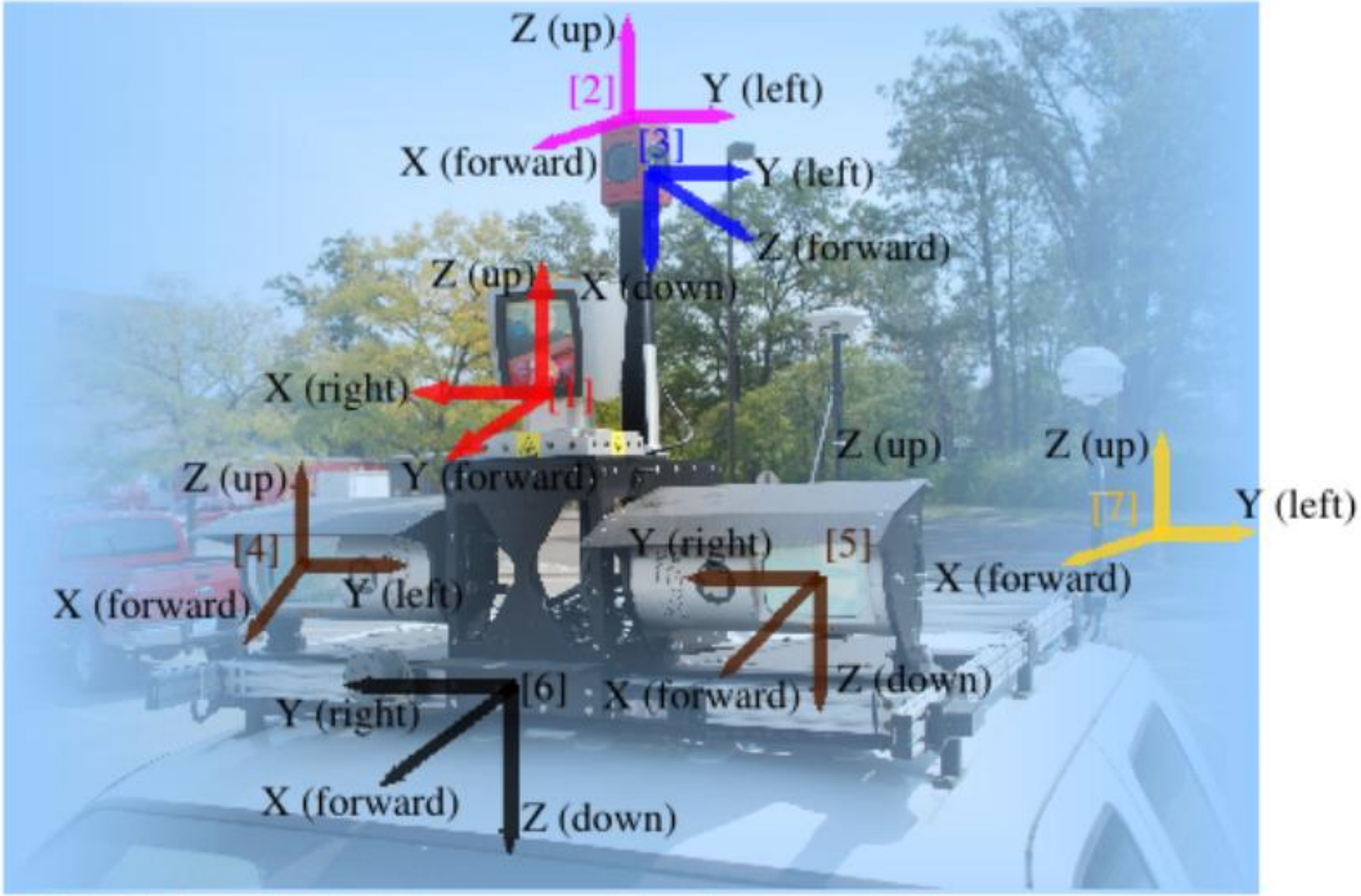
Why do we need to use so many frames?

- Because we want to reason and express quantities relative to their local configuration.

- For example: “grab the bottle behind the cereal bowl”

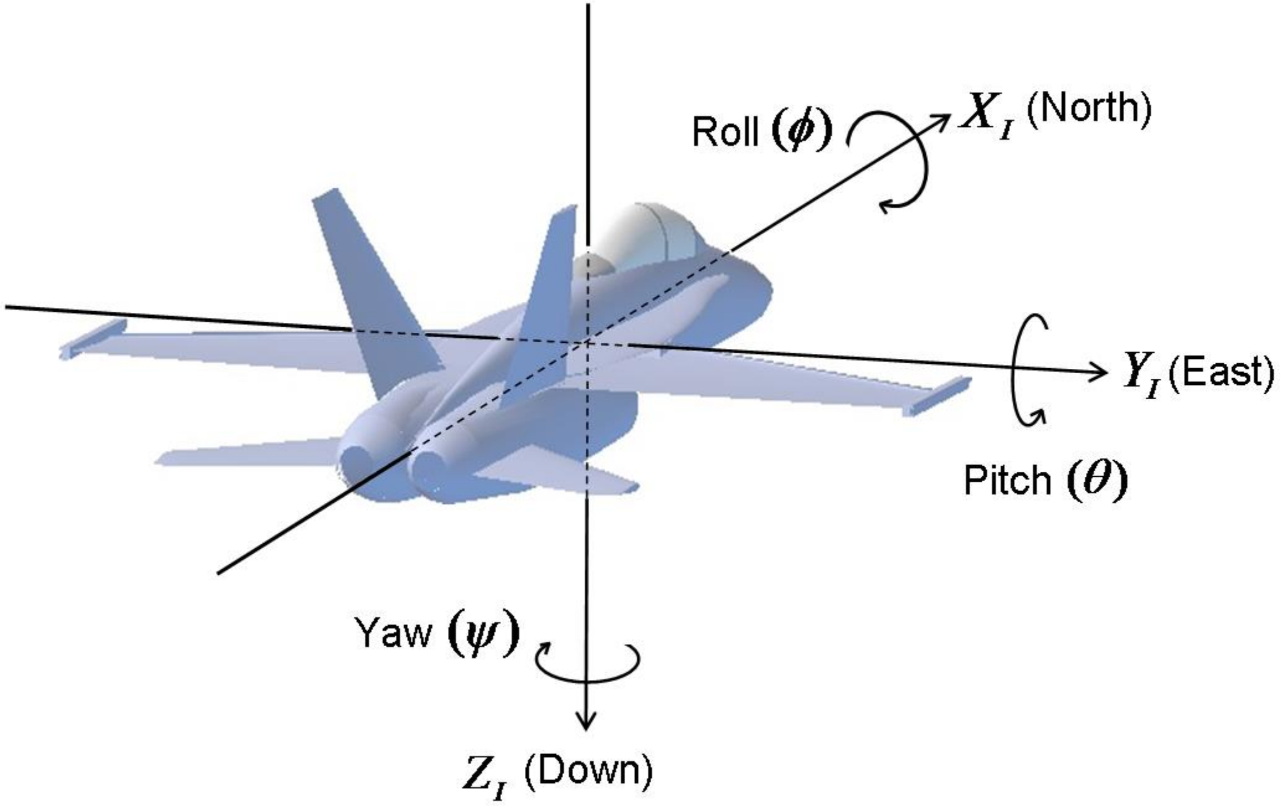
- Many algorithms in this class are mostly about representing frames of reference and reasoning about how to express quantities in one frame to quantities in the other.

The frames of self-driving



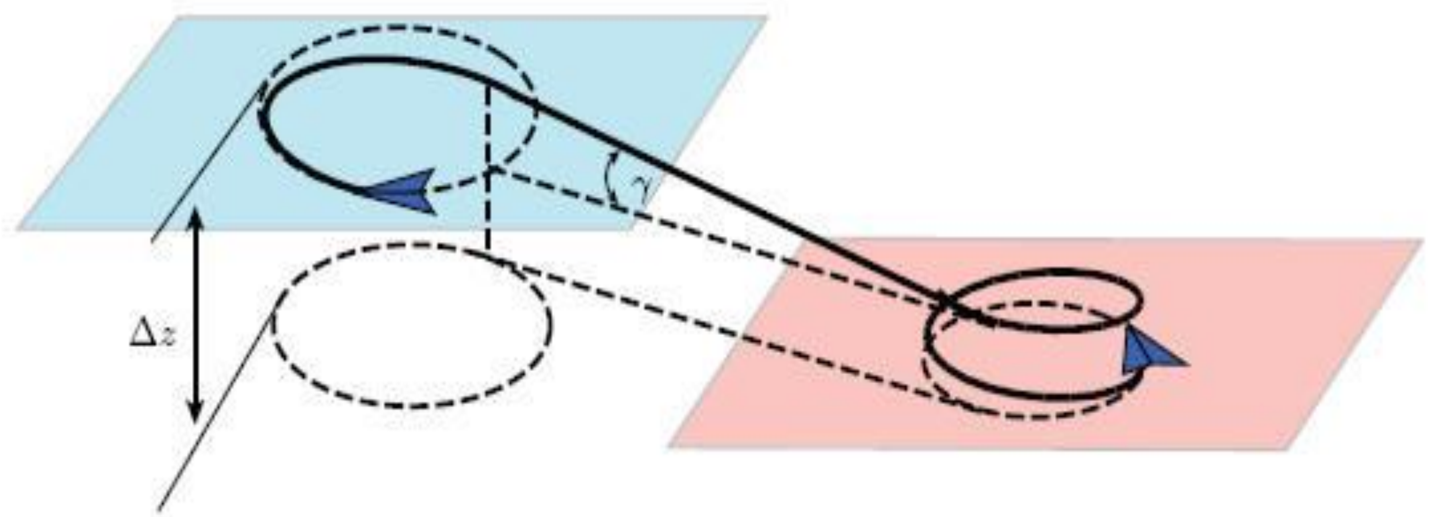
[1] Velodyne, [2] Ladybug3 (actual location: center of camera system),
[3] Ladybug3 Camera 5, [4] Right Riegl, [5] Left Riegl,
[6] Body Frame (actual location: center of rear axle)
[7] Local Frame (Angle between the X-axis and East is known)

Representing Rotations in 3D: Euler Angles



Dubins car ⟶ Dubins airplane in 3D

- Pitch angle ϕ and forward velocity determine descent rate
- Yaw angle θ and forward velocity determine turning rate



(a) The 3D view of the path.

$$
\begin{aligned}
\dot{p}_x &= v_x \cos(\theta)\sin(\phi) \\
\dot{p}_y &= v_x \sin(\theta)\sin(\phi) \\
\dot{p}_z &= v_x \cos(\phi) \\
\dot{\theta} &= \omega_z \\
\dot{\phi} &= \omega_y
\end{aligned}
$$

θ is yaw

ϕ is pitch

Specification ambiguities in Euler Angles

- Need to specify the axes which each angle refers to.

- There are **12 different valid combinations** of fundamental rotations. Here are the possible axes:

- z-x-z, x-y-x, y-z-y, z-y-z, x-z-x, y-x-y
- x-y-z, y-z-x, z-y-x, x-z-y, z-y-x, y-x-z

Specification ambiguities in Euler Angles

- Need to specify the axes which each angle refers to.

- There are **12 different valid combinations** of fundamental rotations. Here are the possible axes:

- z-x-z, x-y-x, y-z-y, z-y-z, x-z-x, y-x-y
- x-y-z, y-z-x, z-y-x, x-z-y, z-y-x, y-x-z

- E.g.: x-y-z rotation with Euler angles (θ, ϕ, ψ) means a rotation: expressed as a sequence of simple rotations $R_x(\theta)R_y(\phi)R_z(\psi)$

Specification ambiguities in Euler Angles

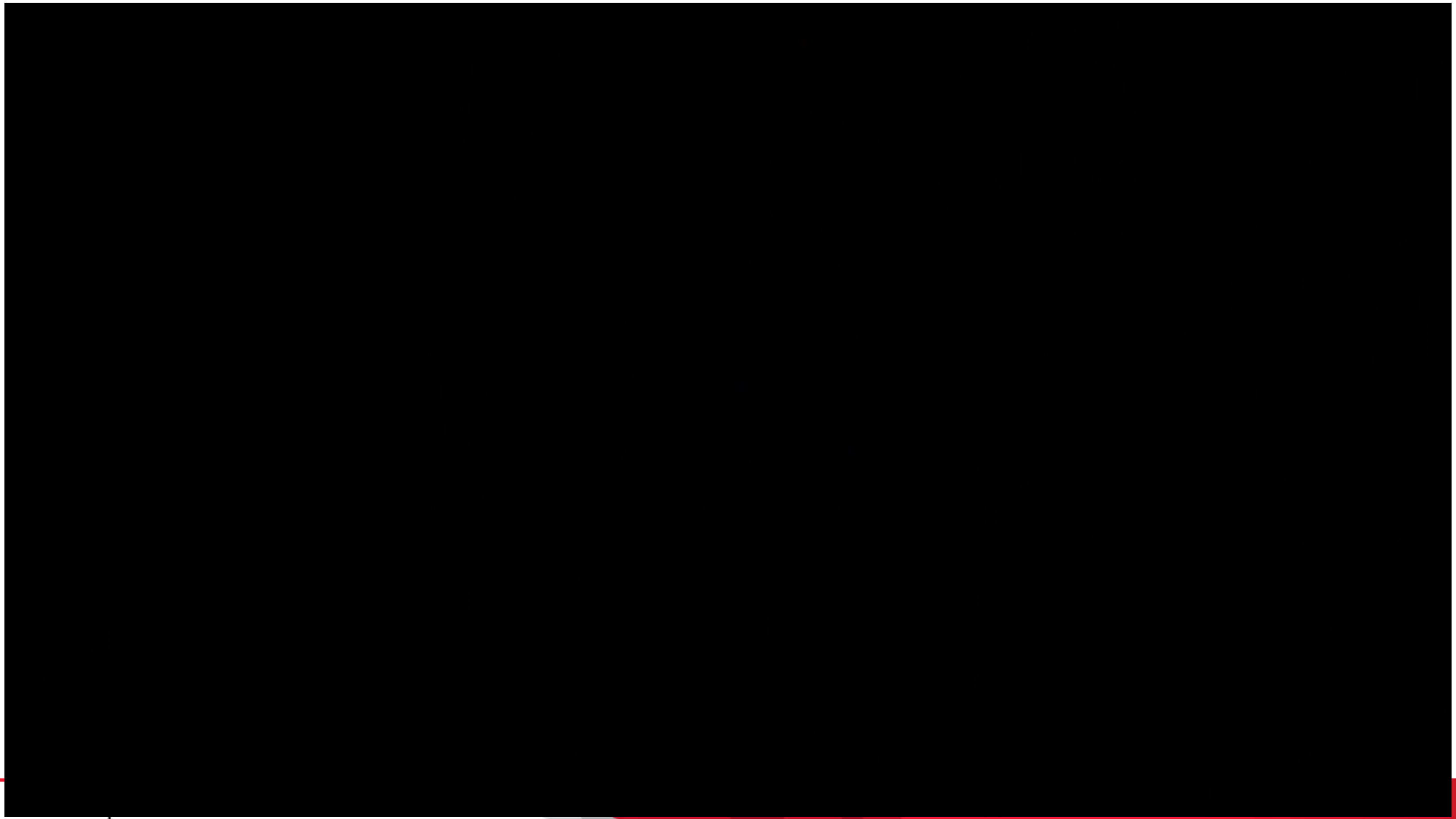
Simple rotations can be counter-clockwise or clockwise.
This gives **another 2 possibilities.**

$$\mathbf{R}_z(\alpha) := \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathbf{C}_z(\alpha) := \begin{bmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

How to handle this horrible situation?

- Use accepted conventions, existing code and libraries. Visualize many rotations and ensure the picture matches your expectations.
- When inventing something new, use extreme documentation.
- When starting with someone else's implementation (such as the provided assignment code), do not assume anything and start by visualizing a wide range of motions.

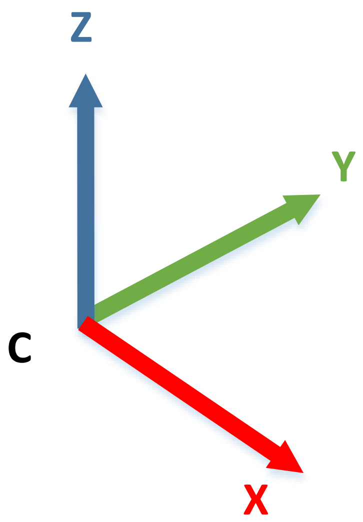
Another problem with Euler angles: Gimbal Lock

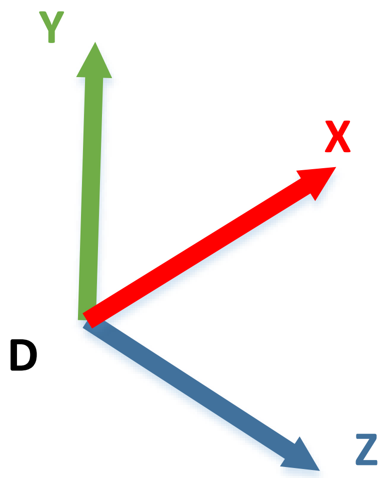


Beyond Euler

- Research papers will often use more complex angle representations:
  - Rotation matrices
  - Rodrigues' axis-angle
  - Quaternions
- This to avoid gimbal lock, imprecise definition and to exploit some particular properties.
- We will not go deep into this in the interest of time

Example: finding a rotation matrix that rotates one vector to another

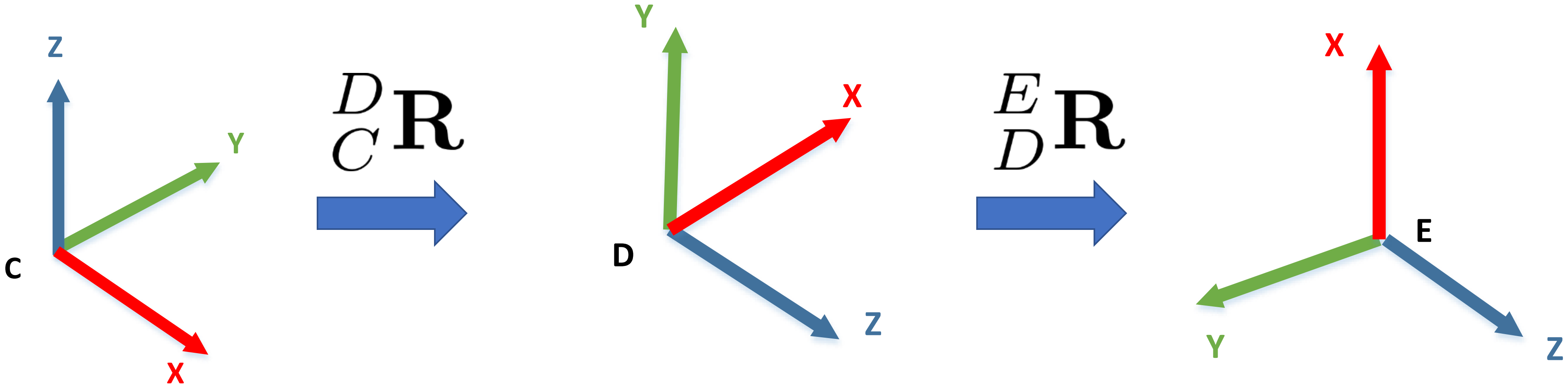




$$ {}^{D}_{C}\mathbf{R} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} $$

This matrix transforms the x-axis of frame C to the z-axis of frame D. Same for y and z axes.

Compound rotations



$$ {}^{E}_{C}\mathbf{R} = {}^{E}_{D}\mathbf{R}\,{}^{D}_{C}\mathbf{R} $$

Passive Dynamics

- Dynamics of systems that operate without drawing (a lot of) energy from a power supply.

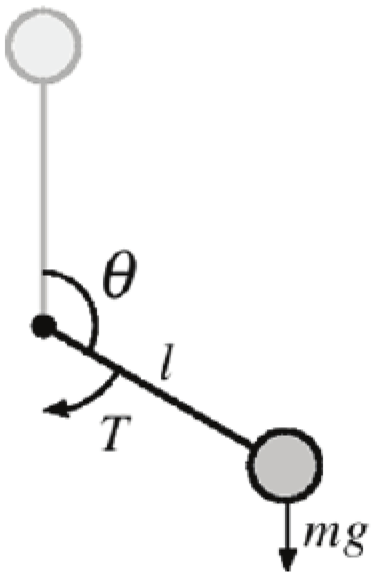
- Interesting because biological locomotion systems are more efficient than current robotic systems.

Dynamics are crucial

Passive Dynamics

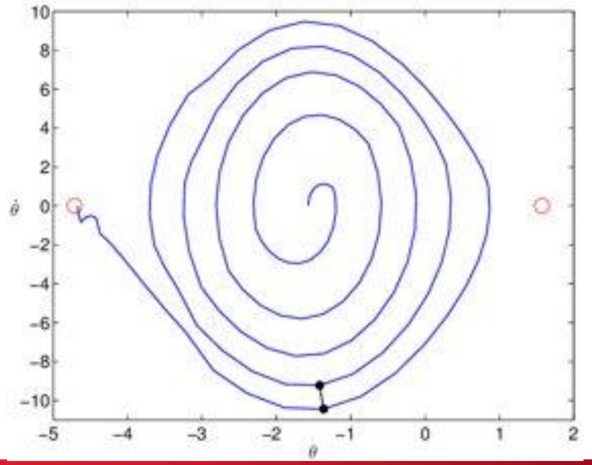
- Dynamics of systems that operate without drawing (a lot of) energy from a power supply.
- Usually propelled by their own weight.
- Interesting because biological locomotion systems are more efficient than current robotic systems.

Important Dynamics Properties



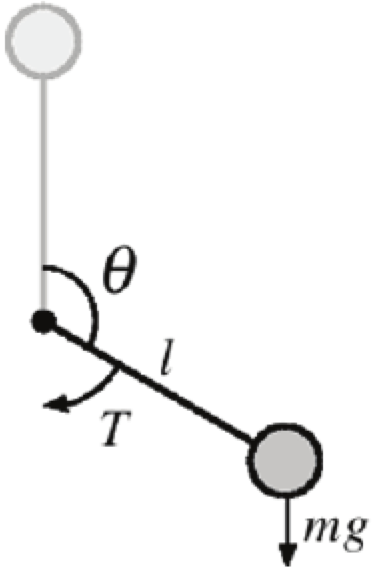
- Stability: does the system diverge or converge over time in a given region?

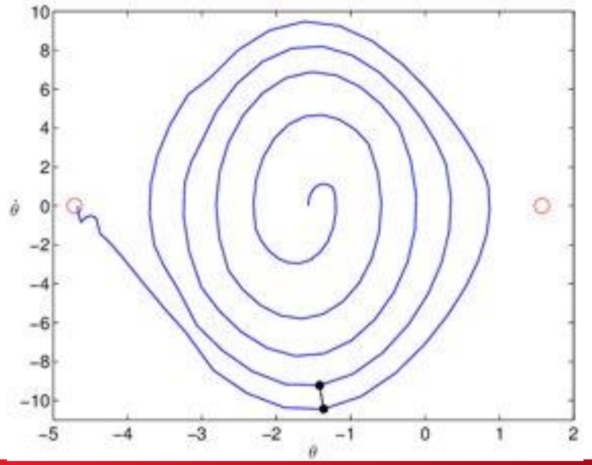
- Controllability: does any action trajectory exist that allows reaching all parts of state space?
  - Without obstacles, this is a property of the dynamics differential equations
  - With obstacles, we can perform computations in the neighborhood



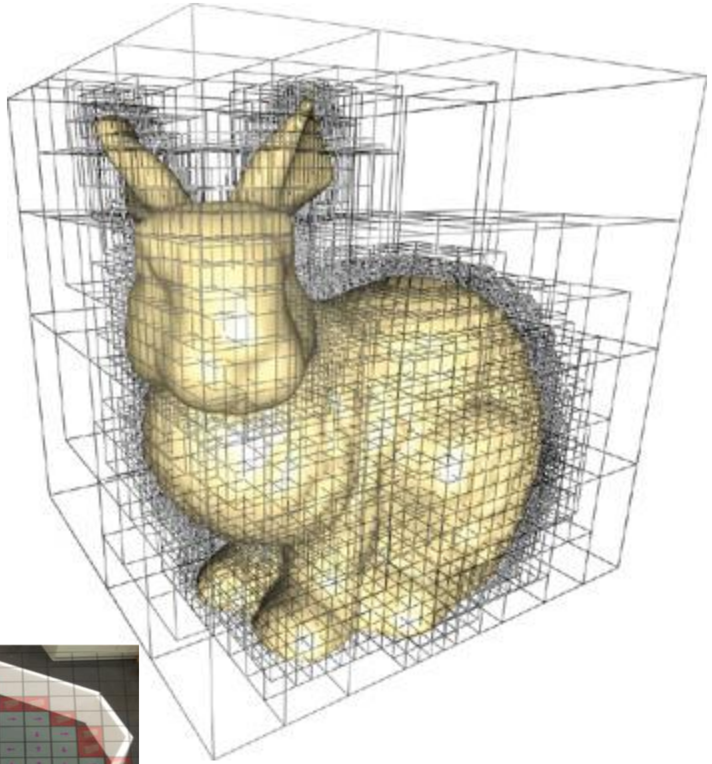
Fully vs Under Actuated

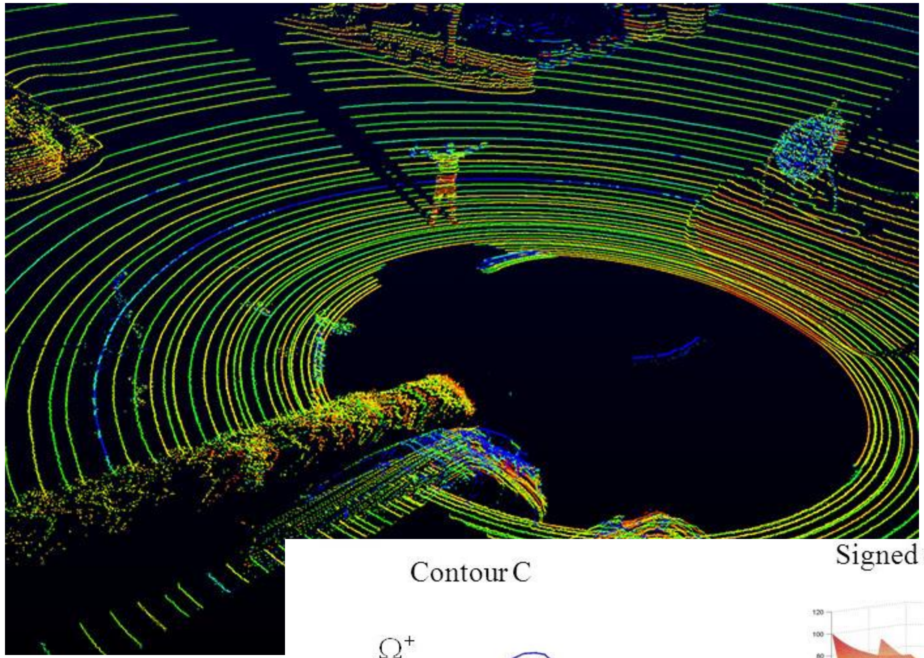
- Somewhat analogous to non-holonomic constraints: an underactuated robot is not able to command an instantaneous acceleration in an arbitrary direction
  - We will see the math detail a bit closer to when we'll use it
- This leads to the need to think farther ahead in the space of actions, and we'll also mostly study these systems:
  - Humans running, jumping
  - Swimming systems
  - Manipulators

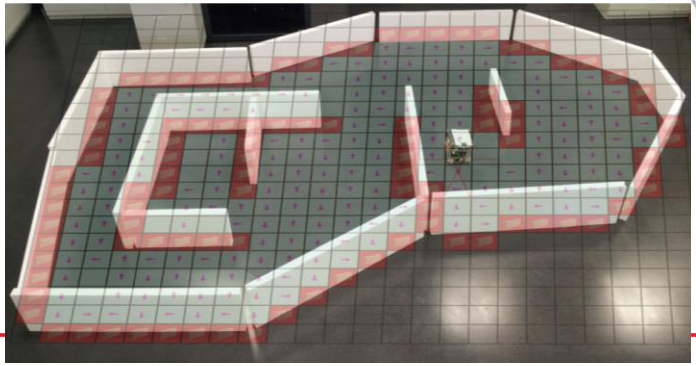


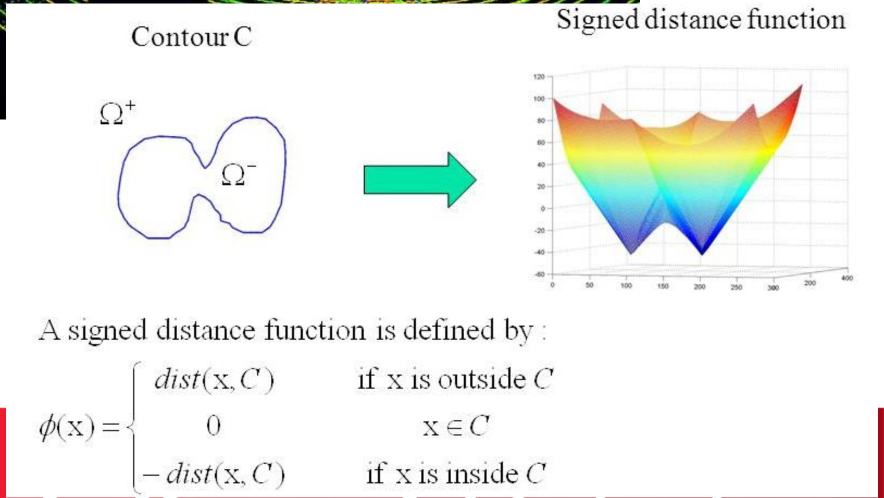


Representing the space around a robot: maps









Categories of maps

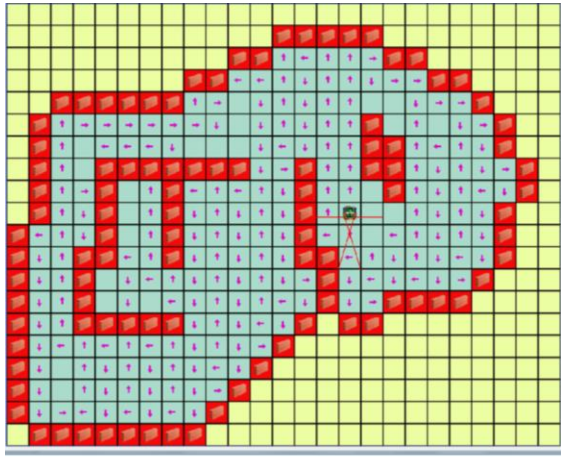
- Metric
  - Map accurately represents lengths and angles
- Topological
  - Map is reduced to a graph representation of the structure of free space
- Topometric
  - Atlas: a combination of local metric maps (nodes) connected via edges
- Sequence of raw time-series observations (e.g. video)
  - No metric or topological information directly represented by the map

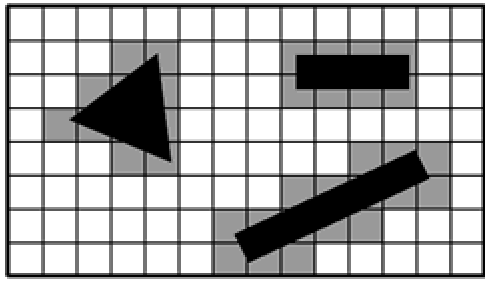
Typical operations on maps

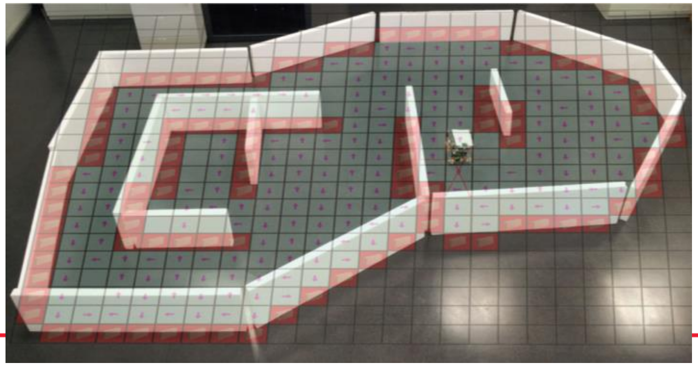
- Distance and direction to closest obstacle
- Collision detection: is a given robot configuration in free space?
- Map merging / alignment
- Occupancy updates
- Raytracing

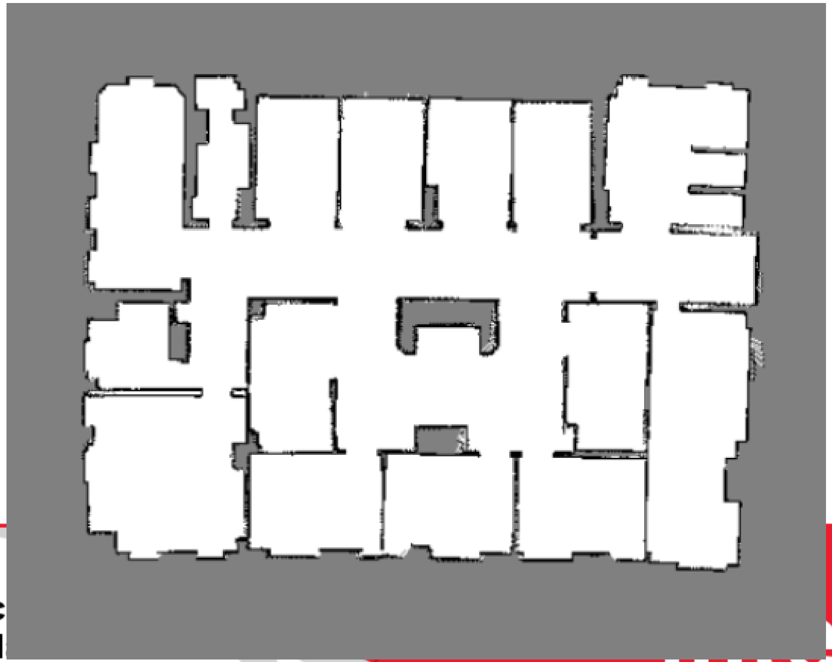
Metric Maps
(primary quantity: spatial position)

Occupancy Grids





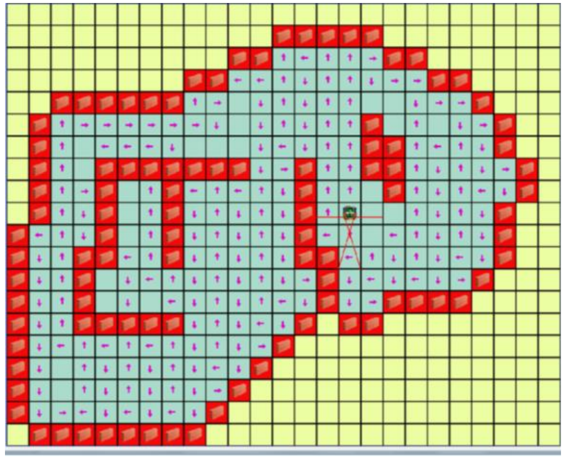


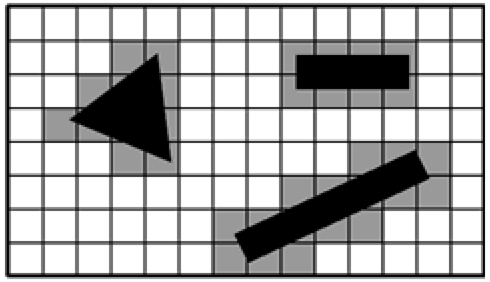


Each cell contains either:

- unknown/unexplored (grey)
- probability of occupation

Occupancy Grids



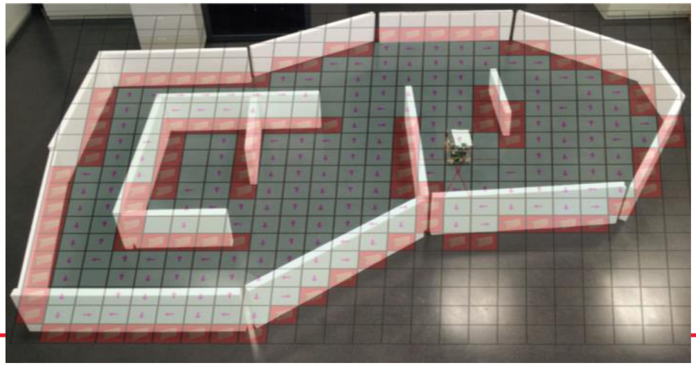


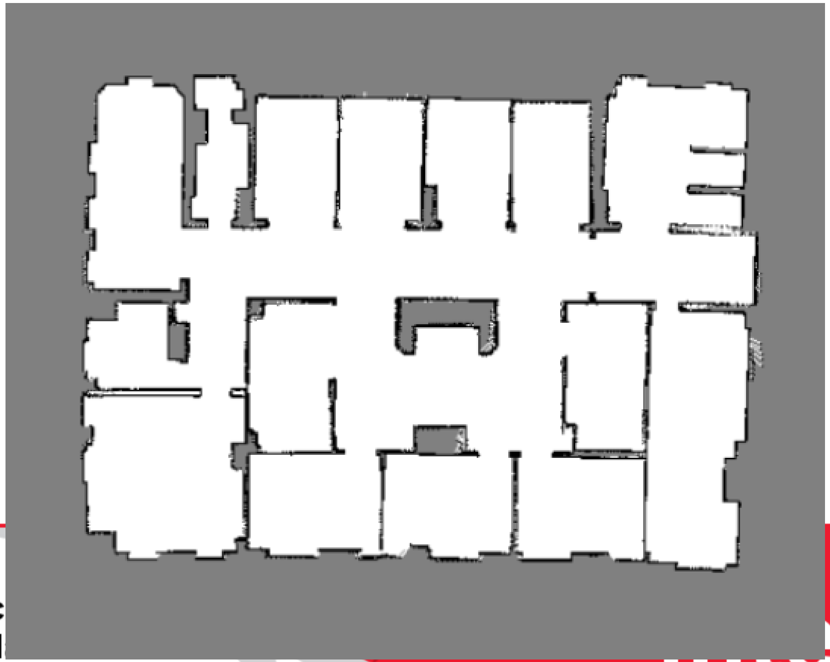
Advantages:

- O(1) occupancy lookup and update
- Supports image operations

Disadvantages:

- Doesn’t scale well in higher dimensions

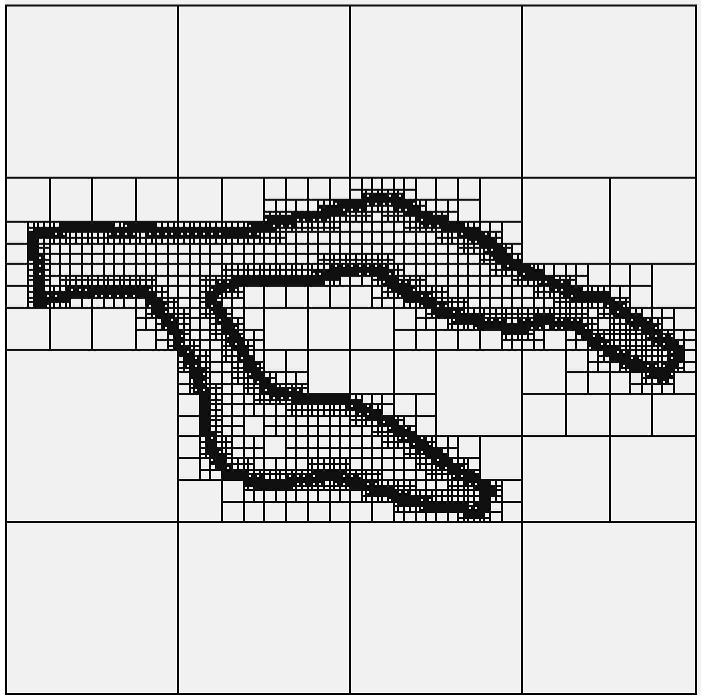
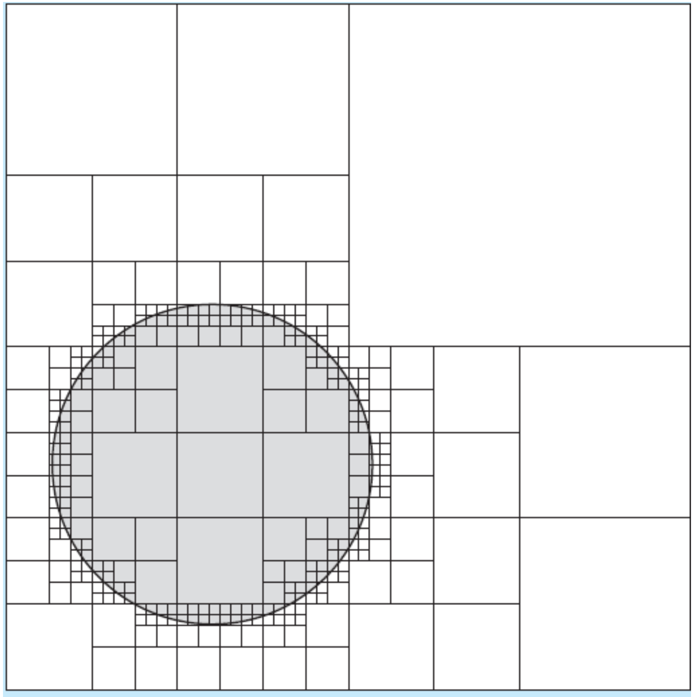
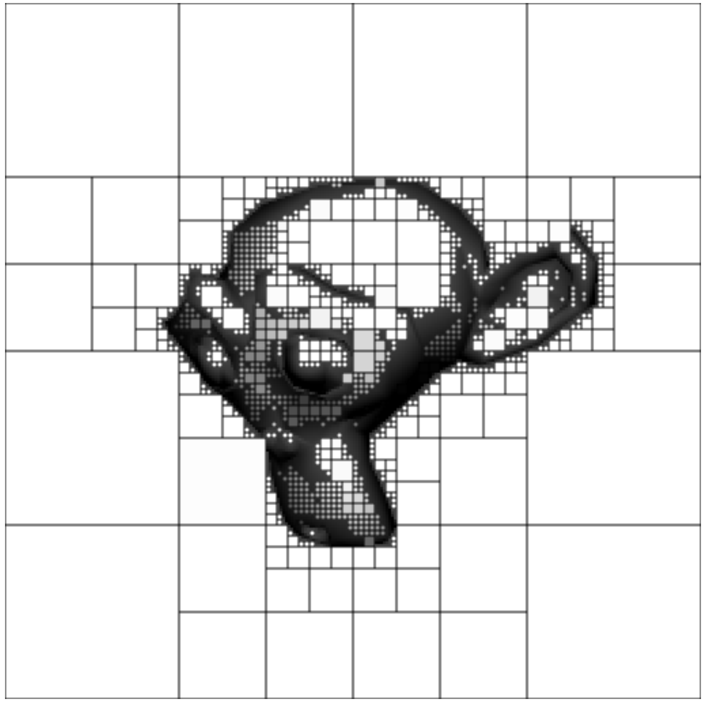




Each cell contains either:

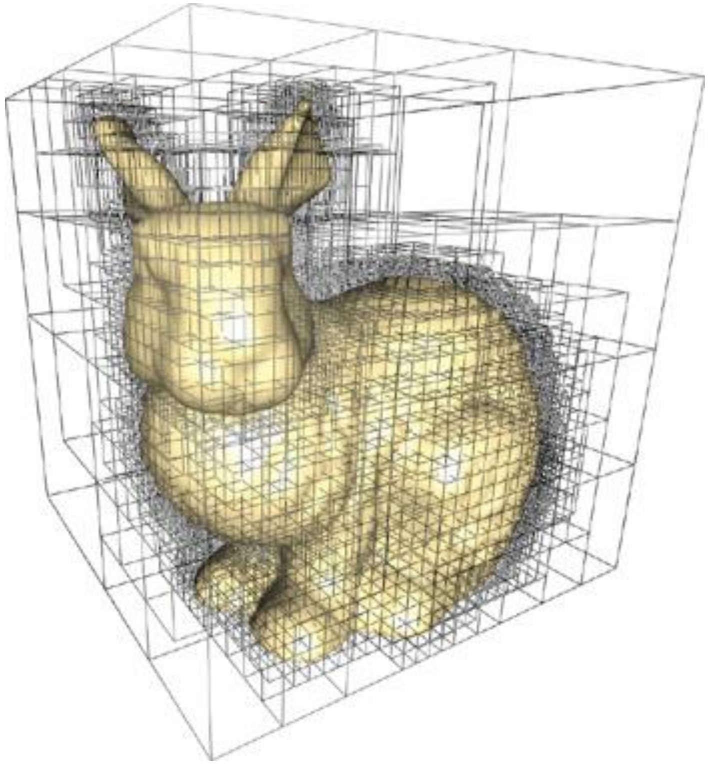
- unknown/unexplored (grey)
- probability of occupation

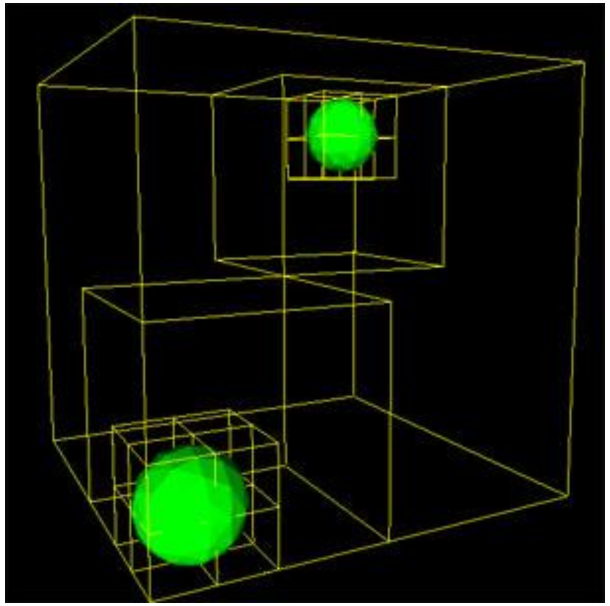
Quadtrees

Each node represents a square. If the node is fully empty or fully occupied it has no children.
If it is partially occupied it has four children. Subdivision stops after some minimal square size.

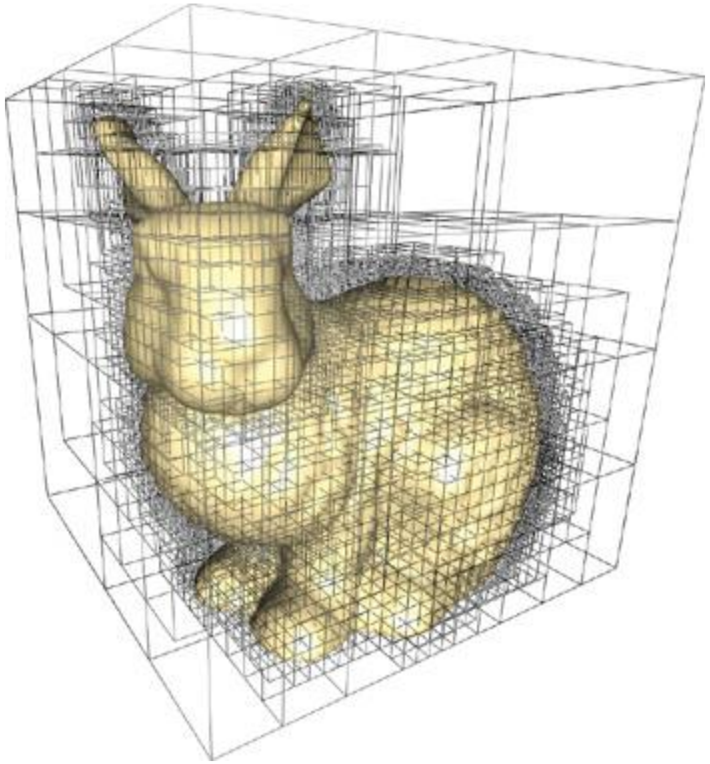
Octrees



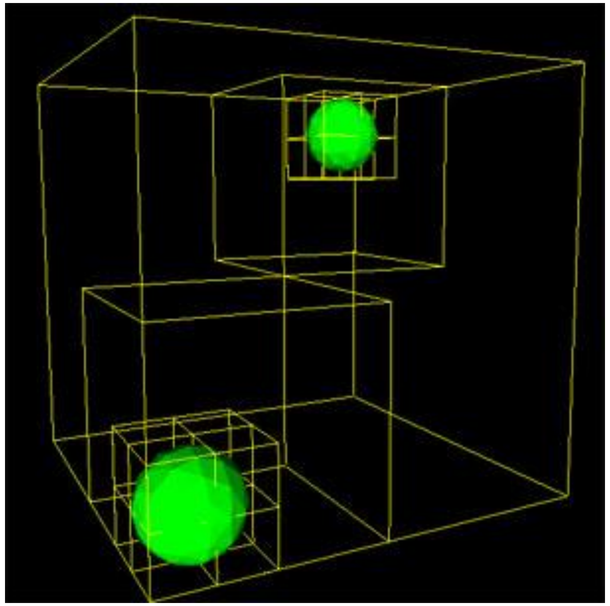


Each node represents a cube. If the node is fully empty or fully occupied it has no children. If it is partially occupied it has eight children. Subdivision stops after some minimal cube size.

Octrees

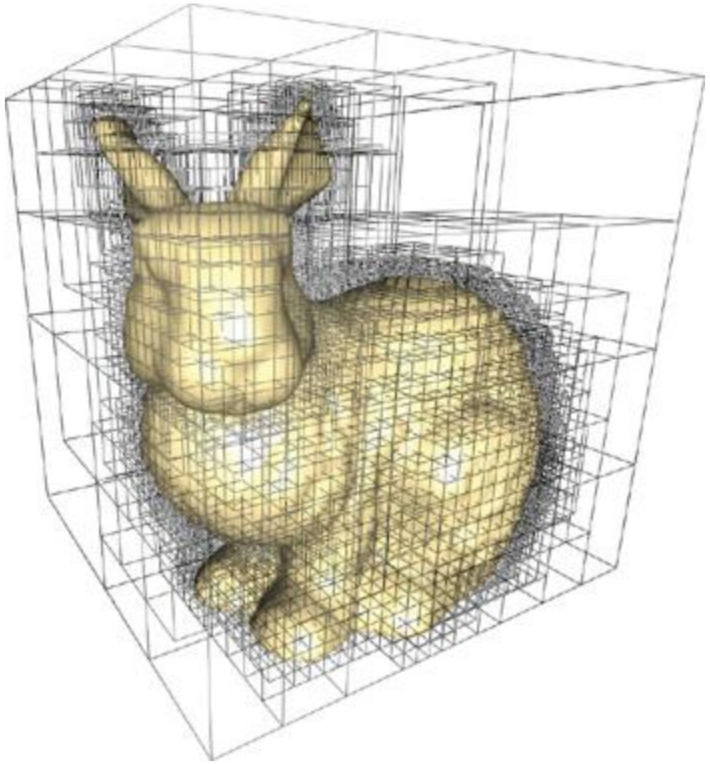


Problem 1: quadtrees and octrees are not balanced trees. So, in the worst case an occupancy query could be O(n) in the number of nodes.



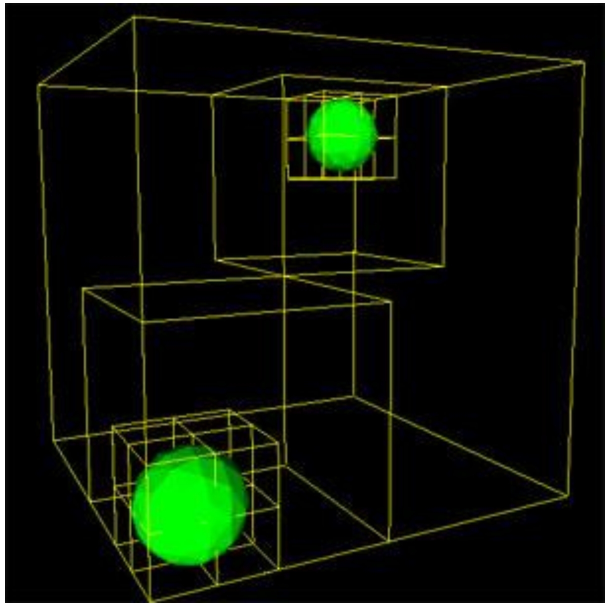
Each node represents a cube. If the node is fully empty or fully occupied it has no children.
If it is partially occupied it has eight children. Subdivision stops after some minimal cube size.

Octrees



Problem 1: quadtrees and octrees are not balanced trees. So, in the worst case an occupancy query could be O(n) in the number of nodes.

Problem 2: quadtrees and octrees are sensitive to small changes in the location of obstacles.



Each node represents a cube. If the node is fully empty or fully occupied it has no children. If it is partially occupied it has eight children. Subdivision stops after some minimal cube size.

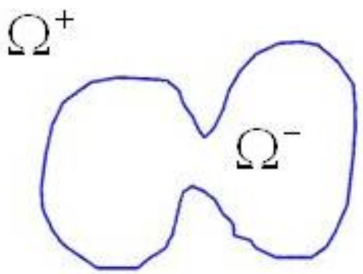
Octree Example: Octomap

Open source
as a ROS package

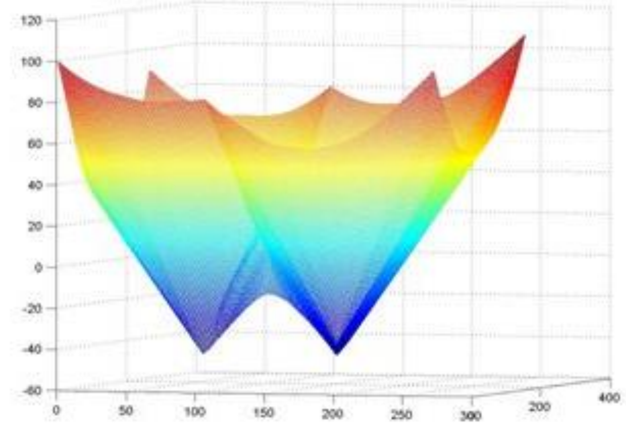
Implicit Surface Definitions: Signed Distance Function

This distance function is defined over any point in 3D space.

Contour C



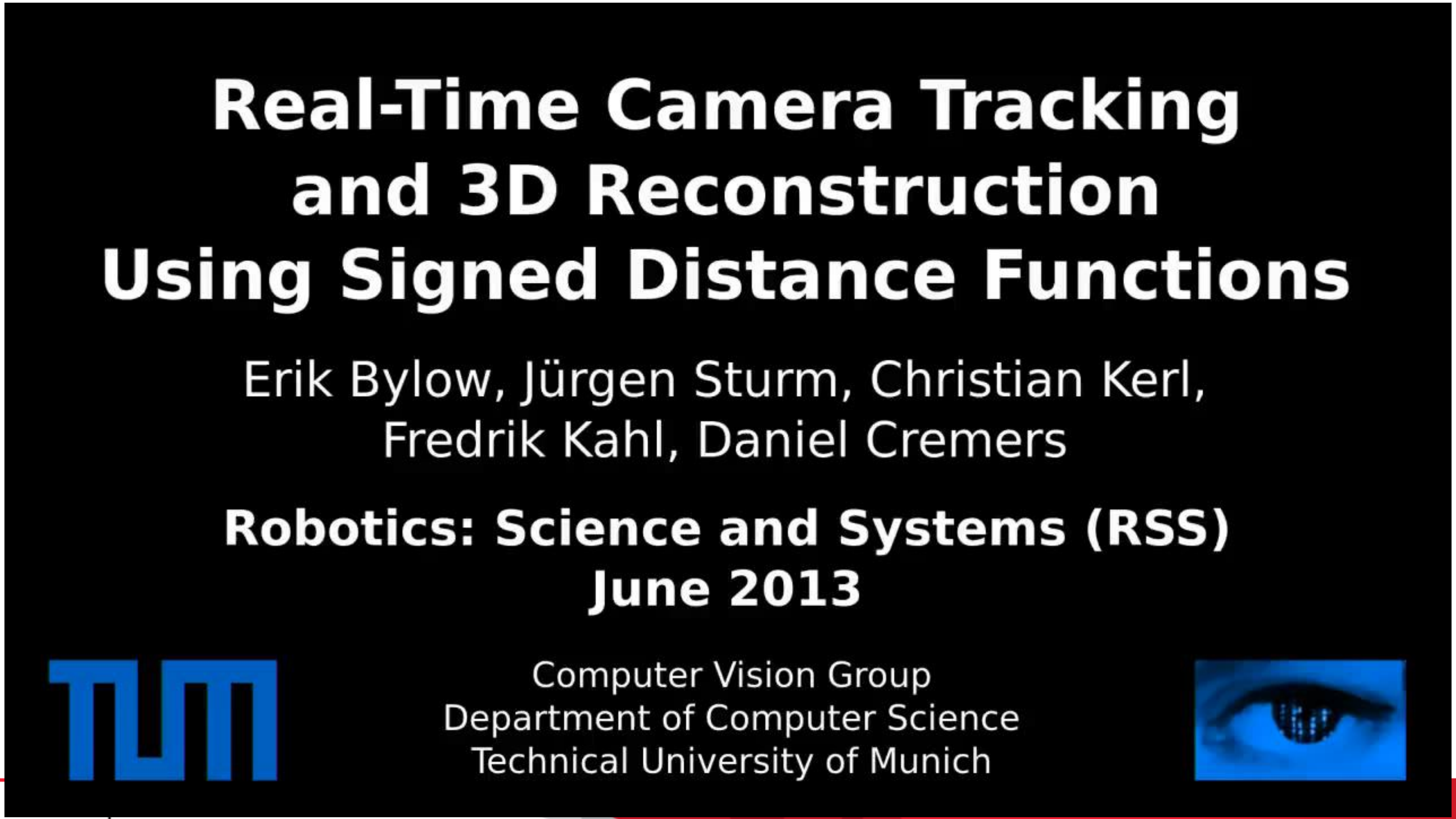
Signed distance function



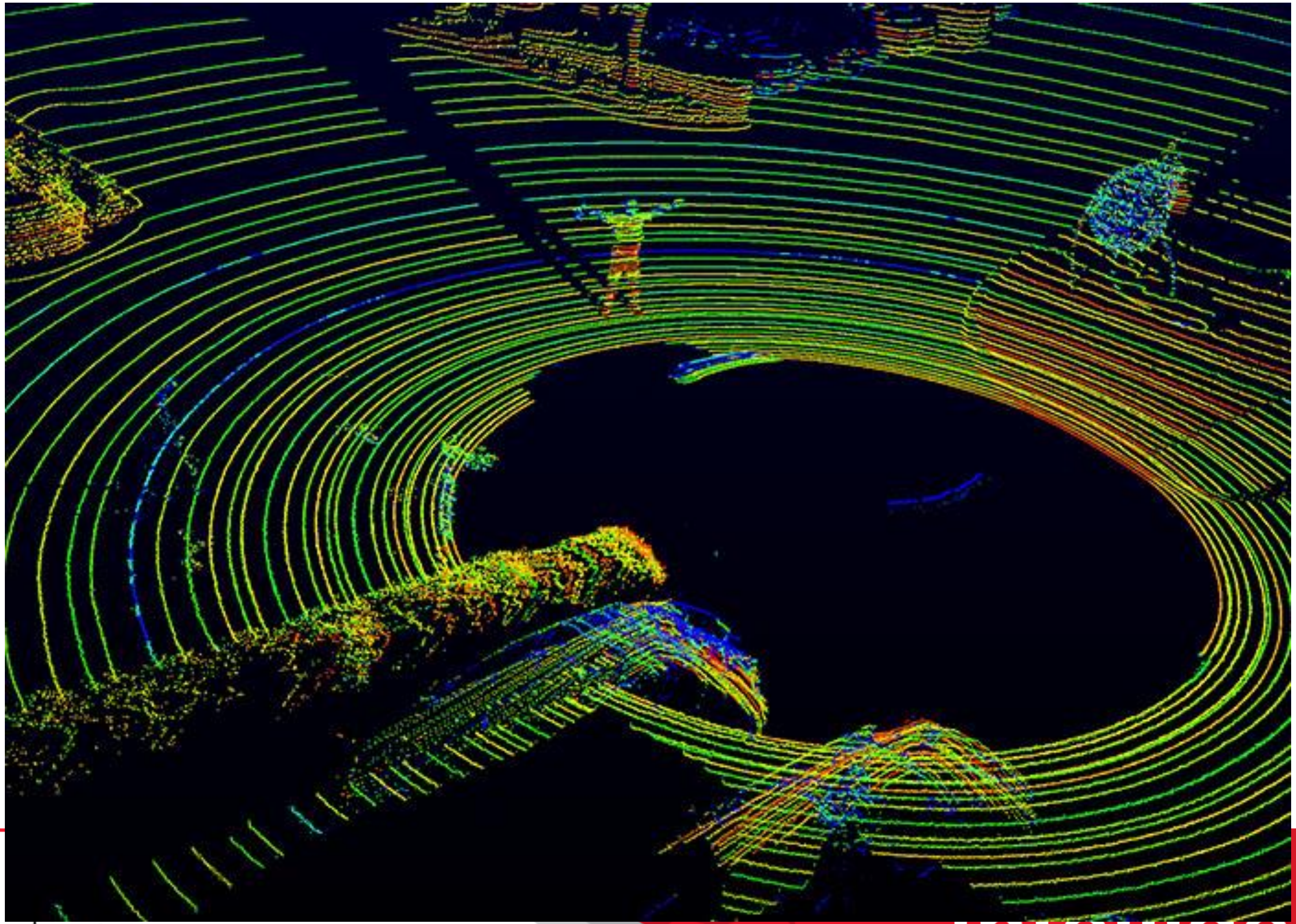
A signed distance function is defined by :

$$\phi(\mathrm{x}) = \begin{cases} dist(\mathrm{x}, C) & \text{if x is outside } C \\ 0 & \mathrm{x} \in C \\ -dist(\mathrm{x}, C) & \text{if x is inside } C \end{cases}$$

SDF Example


Real-Time Camera Tracking
and 3D Reconstruction
Using Signed Distance Functions
Erik Bylow, Jürgen Sturm, Christian Kerl,
Fredrik Kahl, Daniel Cremers
Robotics: Science and Systems (RSS)
June 2013
Computer Vision Group
Department of Computer Science
Technical University of Munich

Pointclouds



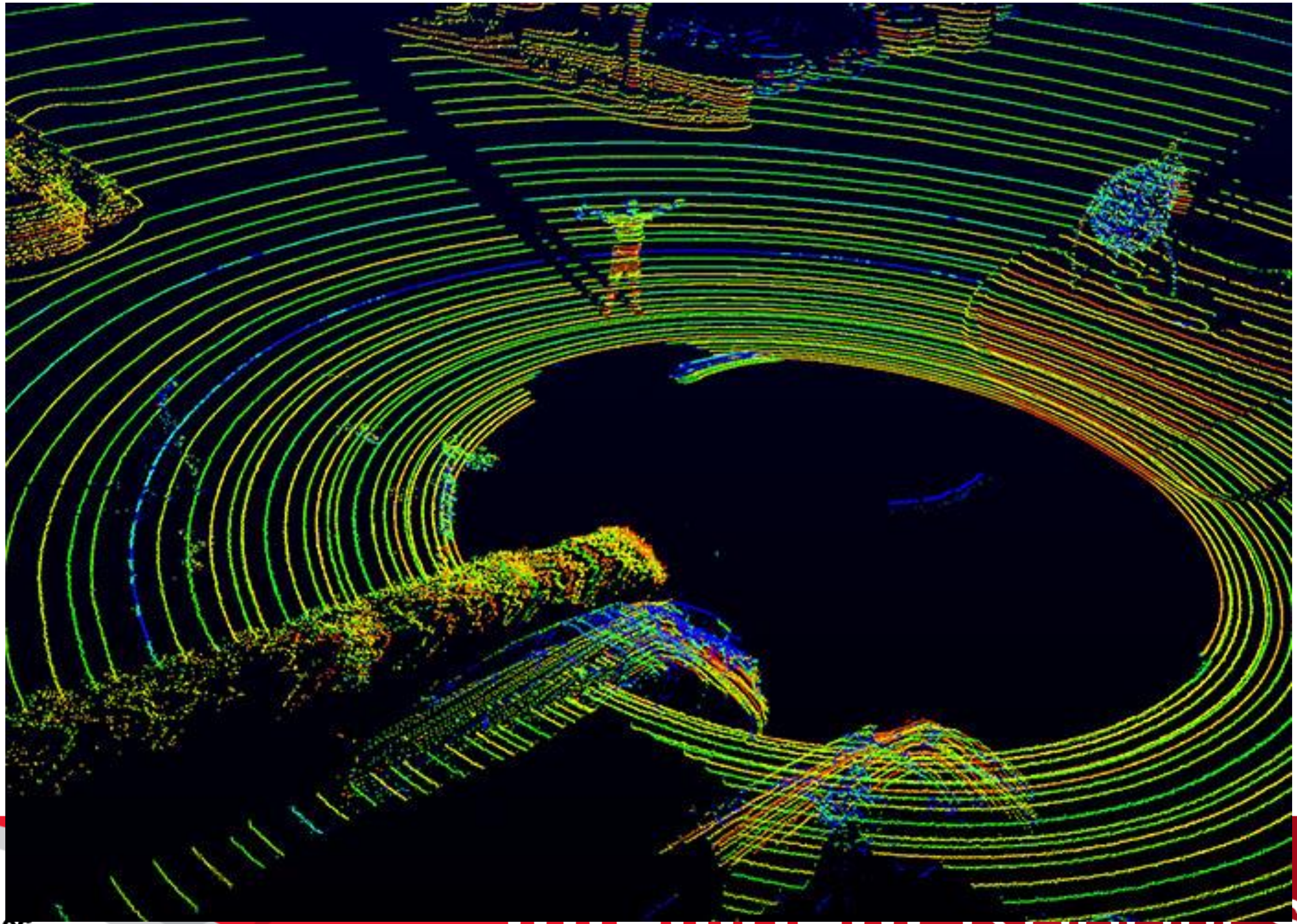
Pointclouds

Advantages:

- can make local changes to the map without affecting the pointcloud globally
- can align pointclouds
- nearest neighbor queries are easy with kd-trees or locality-sensitive hashing

Disadvantages:

- need to segment objects in the map
- raytracing is approximate and nontrivial



The Utility of Models

- "All models are wrong, but some are useful" – George Box (statistician)
- Model: a function that describes a physical phenomenon or a system, i.e. how a set of input variables cause a set of output variables.
- Models are useful if they can predict reality <u>up to some degree</u>.
- Mismatch between model prediction and reality = **error / noise**
- We need to use these models carefully, and mix them with data that can improve performance!

So, where does the data come from?

- A robot's sensors give it feedback on its internal state (proprioception), and allow it to observe the external world (exteroception)

- Like models, no sensor is perfect, but much work goes into making sensor data optimally useful for robotics applications:
  - What a pain… thank goodness we'll see how probability helps here next week!

Types of sensors

- General classification:
  - contact vs. non-contact
  - active vs. passive
  - sampling rate: fast vs. slow
  - local vs. non-local

- General examples:
  - vision
  - laser
  - radar
  - sonar
  - compass, gyroscope, accelerometer
  - touch (tactile)
  - infrared

Sensors

- Devices that can sense and measure physical properties of the environment.

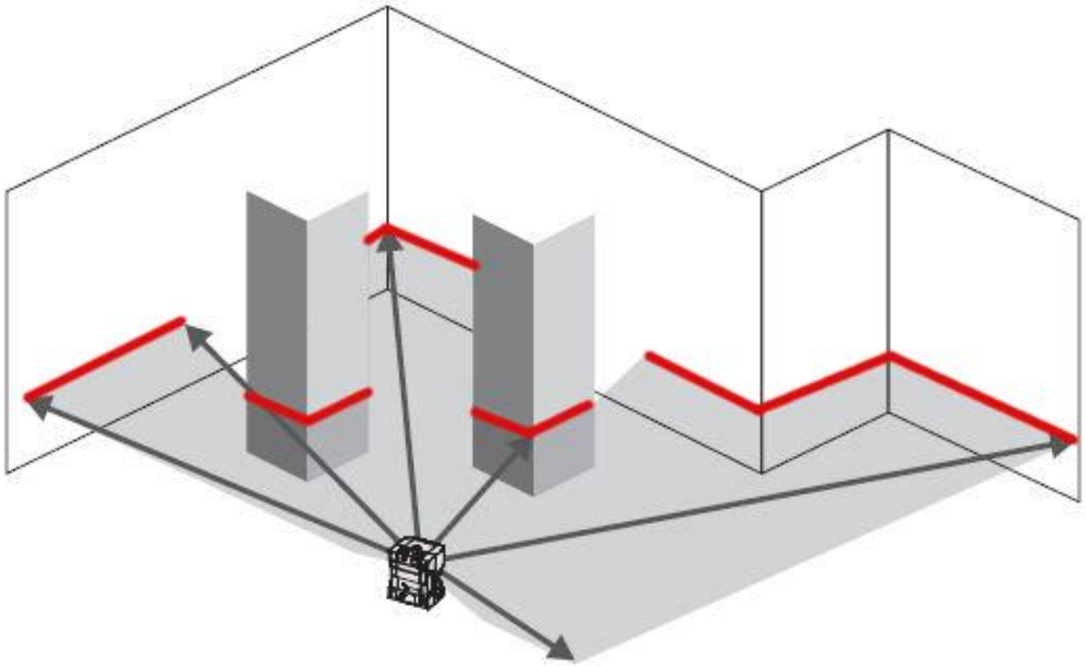
- Key phenomenon is **transduction** (conversion of energy from one form to another). E.g.:
  - Imaging sensors: light to pixel voltages
  - Depth sensors: mechanical pressure to voltage

- Measurements are **noisy**, and difficult to interpret

2D LIDAR (Light detection and ranging)

Produces a scan of 2D points and intensities
- (x,y) in the laser's frame of reference
- Intensity is related to the material of the object that reflects the light

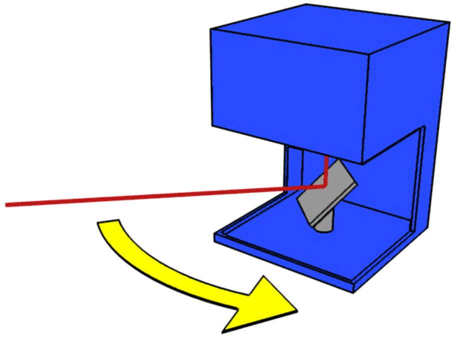
Certain surfaces are problematic for LIDAR: e.g. glass

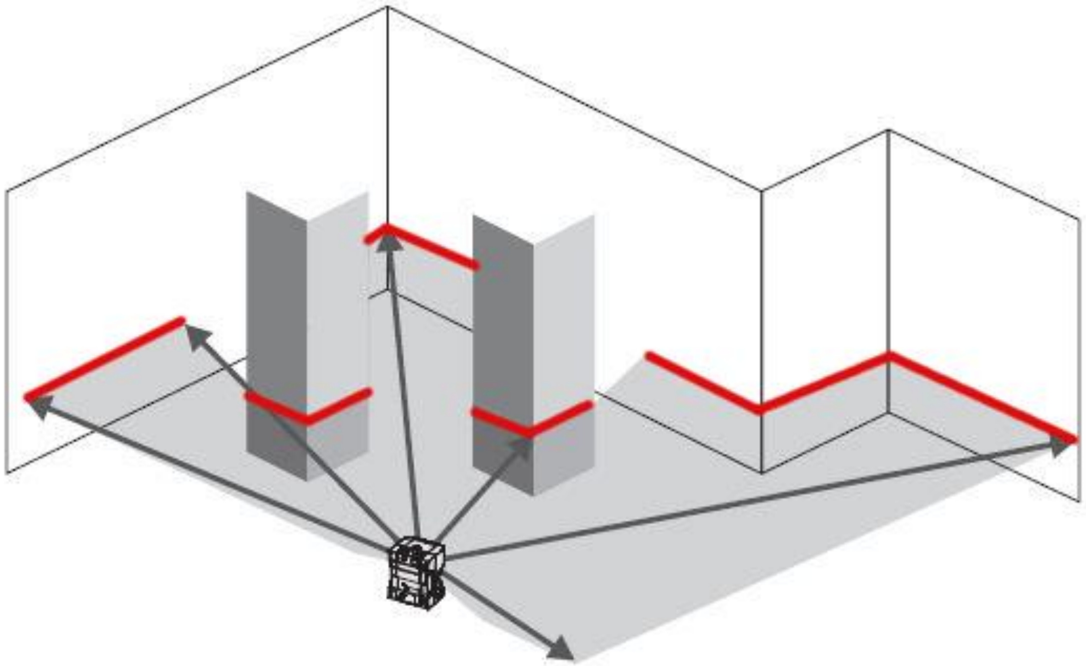


2D LIDAR (Light detection and ranging)

Produces a scan of 2D points and intensities
- (x,y) in the laser's frame of reference
- Intensity is related to the material of the object that reflects the light

Certain surfaces are problematic for LIDAR: e.g. glass



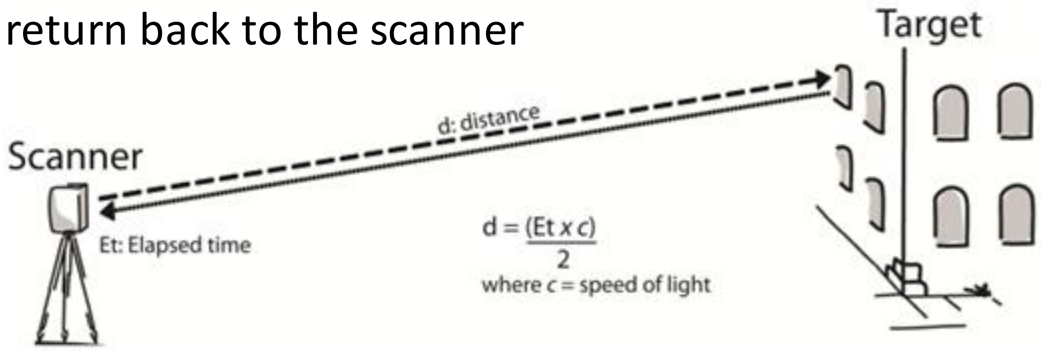


Usually around 1024 points in a single scan.

3D LIDAR (Light detection and ranging)

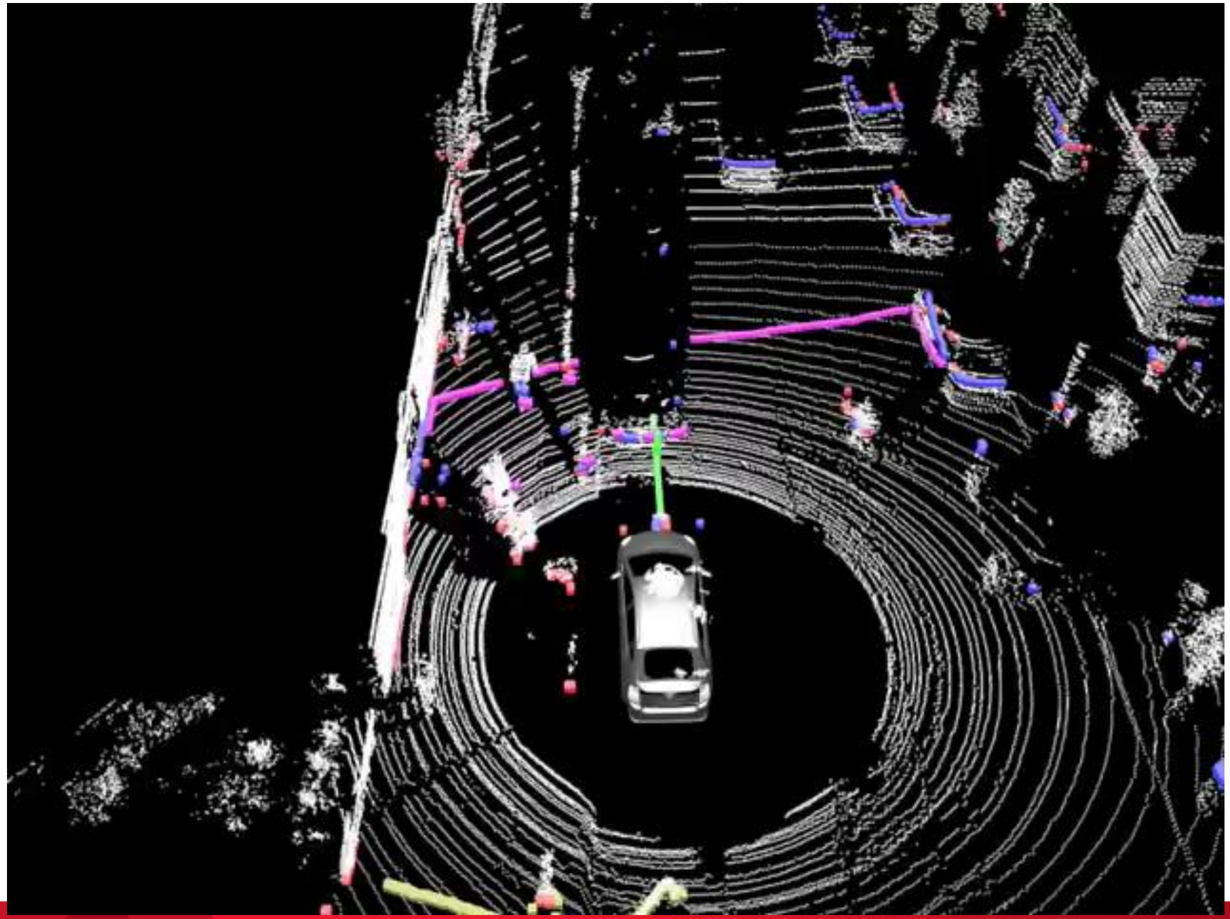
Produces a pointcloud of 3D points and intensities
- (x,y,z) in the laser's frame of reference
- Intensity is related to the material of the object that reflects the light

Works based on time-of-flight for each beam to return back to the scanner



Not very robust to adverse weather conditions: rain, snow, smoke, fog etc.

Usually around 1million points in a single pointcloud



Inertial Sensors

- Gyroscopes, Accelerometers, Magnetometers
- Inertial Measurement Unit (IMU)

- Perhaps the most important sensor for 3D navigation, along with the GPS

- Without IMUs, plane autopilots would be much harder, if not impossible, to build

Beyond the visible spectrum: RGBD cameras

Main ideas:

- Active sensing
- Projector emits infrared light in the scene
- Infrared sensor reads the infrared light
- Deformation of the expected pattern allows computation of the depth

Beyond the visible spectrum: RGBD cameras

Drawbacks:
- Does not work outdoors, sunlight saturates its measurements
- Maximum range is [0.5, 8] meters

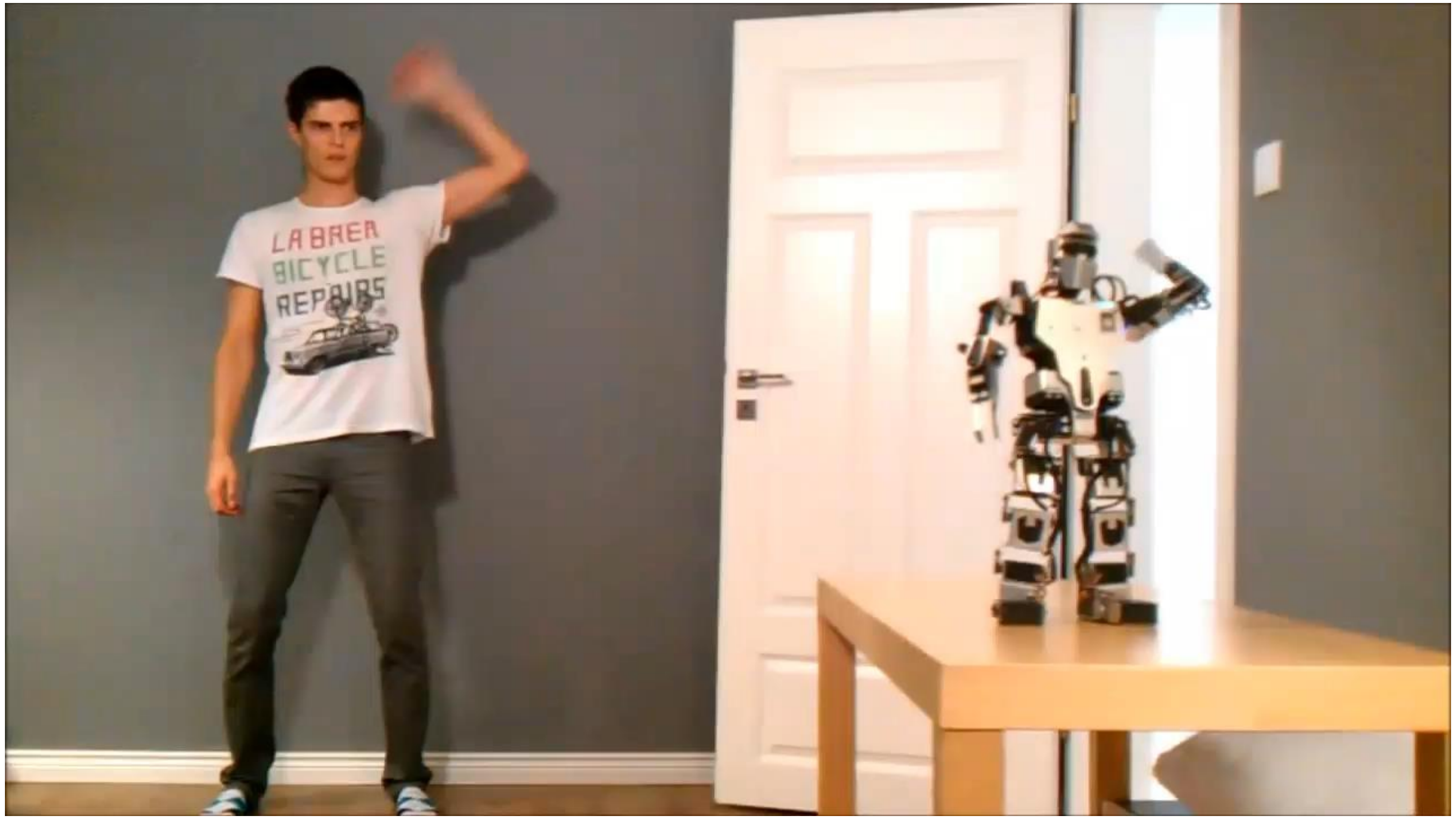
Advantages:
- Real-time depth estimation at 30Hz
- Cheap

Beyond the visible spectrum: RGBD cameras

Enabled a wave of research, applications, and video games, based on real-time skeleton tracking

Beyond the visible spectrum: RGBD cameras

Despite their drawbacks RGBD sensors have been extensively used in robotics.



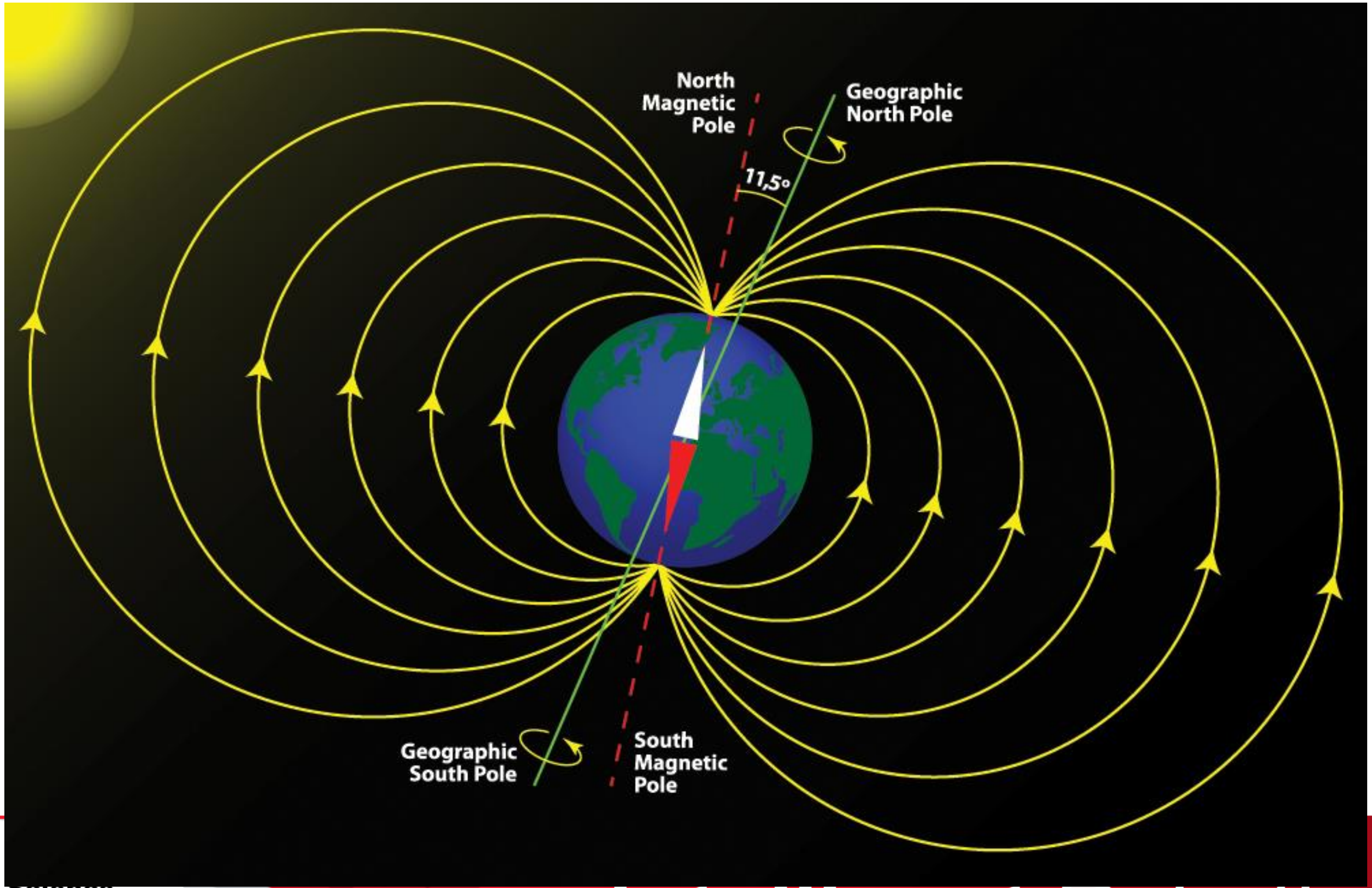
Magnetometers

Drawbacks:

- Needs careful calibration
- Needs to be placed away from moving metal parts, motors

Advantages:

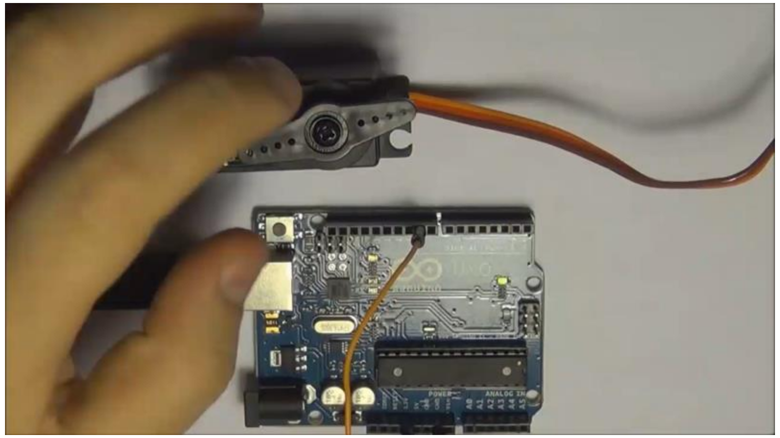
- Can be used as a compass for absolute heading



Actuators are also a form of sensors...

DC (direct current) motor

They turn continuously at high RPM (revolutions per minute) when voltage is applied. Used in quadrotors and planes, model cars etc.

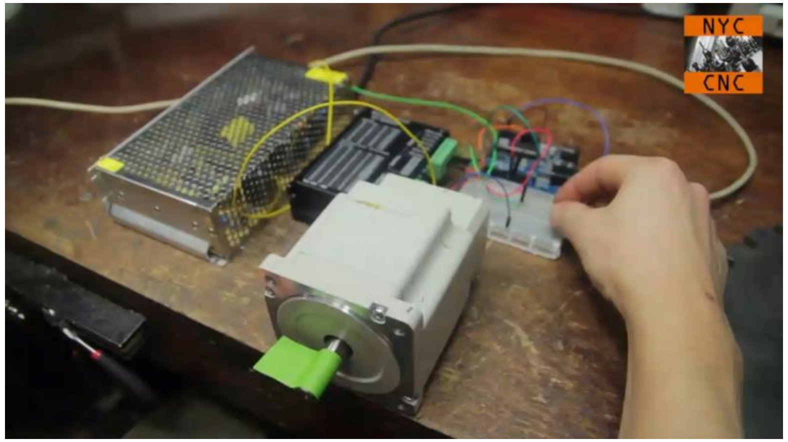


Servo motor

Usually includes: DC motor, gears, control circuit, position feedback

Precise control without free rotation (e.g. robot arms, boat rudders)

Limited turning range: 180 degrees



Stepper motor

Positioning feedback and no positioning errors.

Rotates by a predefined step angle.

Requires external control circuit.

Precise control without free rotation.

Constant holding torque without powering the motor (good for robot arms or weight-carrying systems).

Now for the algorithms!

- Estimation: obtain precise model from imprecise sensing
- Planning: find a sequence of states that achieves a goal
- Control: compute the actions to follow the plan
- Decision making under uncertainty:
  - Acting optimally regardless of noise
  - Acting for the explicit purpose of increasing certainty
  - Acting deceptively, to decrease the certainty of a "pursuer"
  - Acting conservatively to avoid the worse case

What comes next?

- We will use models of motion and perception to estimate a robot's position with uncertainty:
  - $p(x_t \,|x_{t-1}, u_{t-1})$ is a generative model of a robot's one-step motion
  - $p(z_t \,|x_t, m)$ is a generative model of a robot's current sensing

- We can use these models in countless places throughout robotics, but the simplest and most well-studied is localization: "Where am I now?"
  - Infer $p(x_t \,|z_{1\ldots t}, m)$

- We'll start on how to do this next time.

Readings related to this material

- Probabilistic Robotics: Intro, Sensors, Bayesian filtering
- Planning Algorithms: Nice job of filling in the math, for example:
  - http://msl.cs.uiuc.edu/planning/node809.html