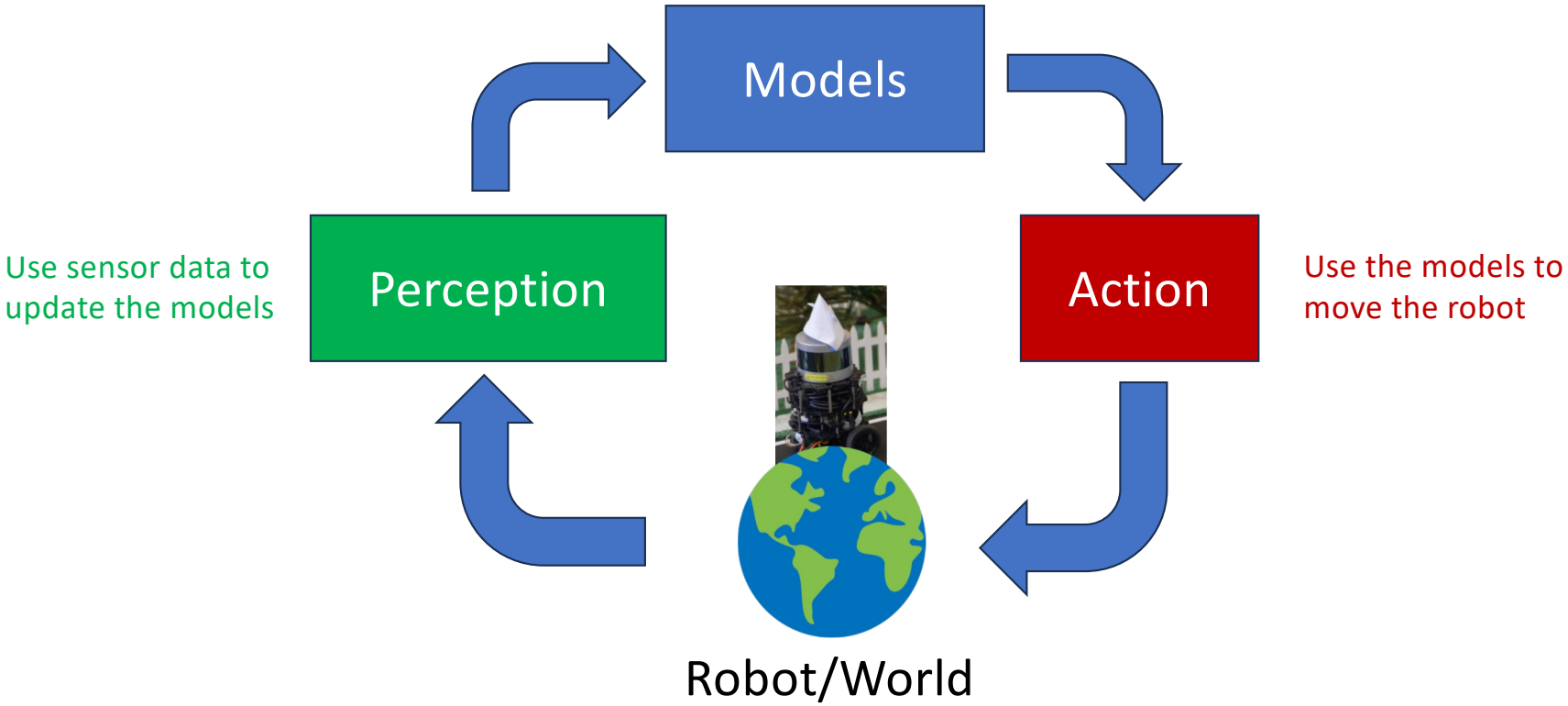
Principles of Robot Autonomy I

Robotic sensors and point cloud alignment

Logistics

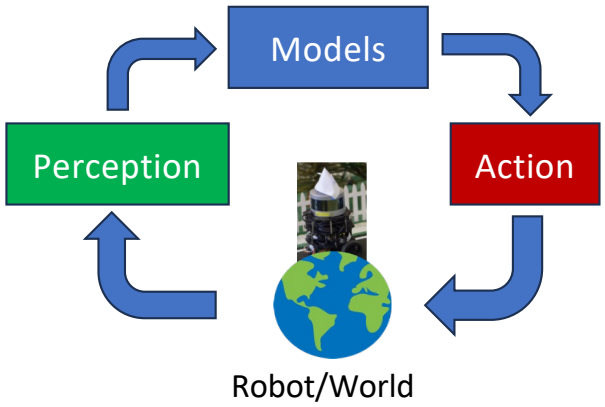
- Homework 2: due **Friday, Oct 17, at 5pm**.
- Homework 3: out Thurs, Oct 16, due Tues, Oct 28
- Midterm window: Wed, Oct 29, 5pm – Fri, Oct 31
  - Take home, 48 hour window
  - Check out exam on gradescope
  - You will have personal 5 hour time slot
  - Open notes, book, HW solutions
  - No internet, no GenAI, no working with others
- Lecture 7:
  - Differential flatness trajectory planning
  - Robot sensors, proprioceptive and exteroceptive
  - Point cloud alignment

Full Stack Autonomy: the Perception-Action Loop



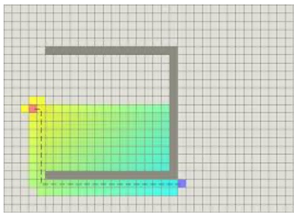
Action: Planning and Control Stack

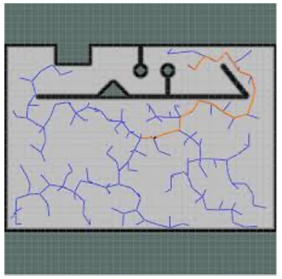
Use the models to move the robot



Path planning:

- A* planning on a graph
- RRT, RRT*



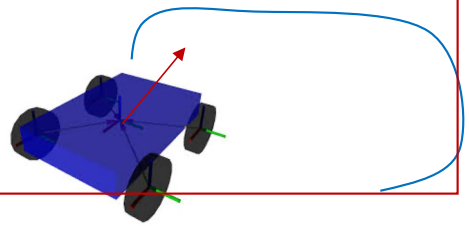


Trajectory optimization:

- Optimal control
- Numerical solvers

Trajectory following:

- Kinematic/dynamic models
- PID control
- LQR, gain scheduled LQR



Differential flatness

Differential flatness: A nonlinear system $\dot{\mathbf{x}} = \boldsymbol{f}(\mathbf{x}, \mathbf{u})$ is differentially flat with flat outputs $\mathbf{z}$ if there exist functions β and γ such that

$$\mathbf{x} = \beta(\mathbf{z}, \dot{\mathbf{z}}, \ldots, \mathbf{z}^{(q)})$$

$$\mathbf{u} = \gamma(\mathbf{z}, \dot{\mathbf{z}}, \ldots, \mathbf{z}^{(q)})$$

and a function α such that

$$\mathbf{z} = \alpha(\mathbf{x}, \mathbf{u}, \ldots, \mathbf{u}^{(p)})$$

In words, a system is differentially flat if we can find a set of outputs (equal in number to the number of inputs) such that all states and inputs can be determined from these outputs *without integration*

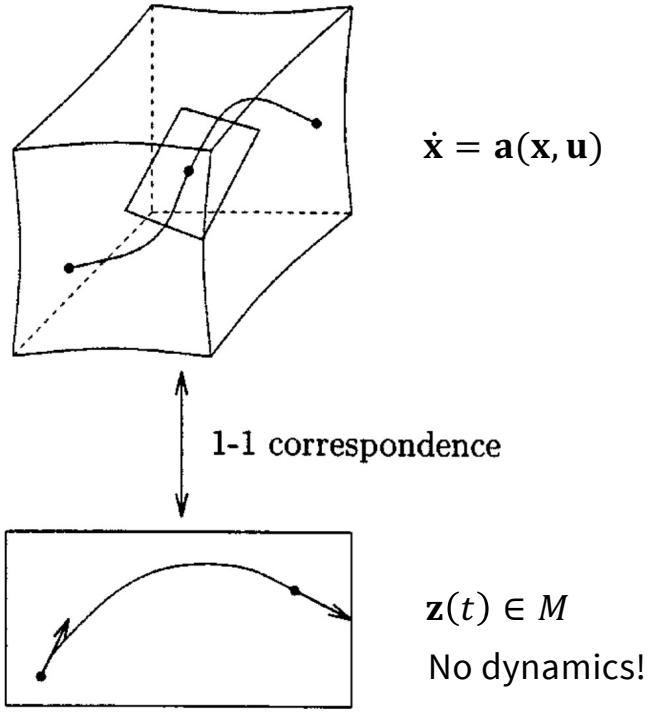
Differential flatness

- Implication for trajectory generation: to every sufficiently differentiable curve $t \to \mathbf{z}(t)$, there is a corresponding trajectory

$$t \to \begin{pmatrix} \mathbf{x}(t) \\ \mathbf{u}(t) \end{pmatrix} = \begin{pmatrix} \beta(\mathbf{z}(t), \dot{\mathbf{z}}(t), \dots, \mathbf{z}^{(q)}(t)) \\ \gamma(\mathbf{z}(t), \dot{\mathbf{z}}(t), \dots, \mathbf{z}^{(q)}(t)) \end{pmatrix}$$

that identically satisfies the system equations

- E.g. The Ackermann steering car, differential drive robot, and quadrotor dynamics are all differentially flat



From Nieuwstadt, Murray. 1998.

Practical implications

This leads to a simple, yet effective strategy for trajectory generation

1. Find the initial and final conditions for the flat output:

Given	**Find**
$(t_0, \mathbf{x}(t_0), \mathbf{u}(t_0))$	$(\mathbf{z}(t_0), \dot{\mathbf{z}}(t_0), \dots, \mathbf{z}^{(q)}(t_0))$
$(t_f, \mathbf{x}(t_f), \mathbf{u}(t_f))$	$(\mathbf{z}(t_f), \dot{\mathbf{z}}(t_f), \dots, \mathbf{z}^{(q)}(t_f))$

2. Build a smooth curve $t \to \mathbf{z}(t)$ for $t \in [t_0, t_f]$, e.g., with splines to fit A* waypoints
3. Compute the corresponding state and control trajectory

$$\mathbf{x} = \beta(\mathbf{z}, \dot{\mathbf{z}}, \dots, \mathbf{z}^{(q)}) \qquad \mathbf{u} = \gamma(\mathbf{z}, \dot{\mathbf{z}}, \dots, \mathbf{z}^{(q)})$$

How do we get the smooth trajectory? Splines!

- As we saw last time
- We can parameterize the flat output trajectory using a set of smooth basis functions $\psi_i(t)$

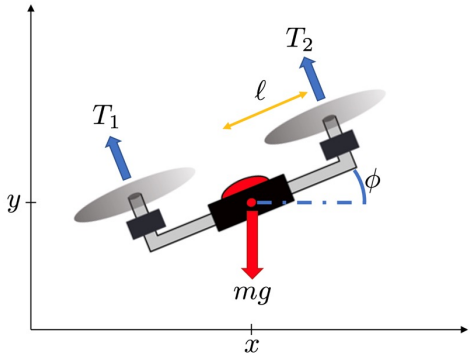
$$z_j(t) = \sum_{i=1}^{N} \alpha_i^{[j]} \psi_i(t)$$

- and then solve

$$\begin{bmatrix} \psi_1(t_0) & \psi_2(t_0) & \dots & \psi_N(t_0) \\ \dot{\psi}_1(t_0) & \dot{\psi}_2(t_0) & \dots & \dot{\psi}_N(t_0) \\ \vdots & \vdots & & \vdots \\ \psi_1^{(q)}(t_0) & \psi_2^{(q)}(t_0) & \dots & \psi_N^{(q)}(t_0) \\ \psi_1(t_f) & \psi_2(t_f) & \dots & \psi_N(t_f) \\ \dot{\psi}_1(t_f) & \dot{\psi}_2(t_f) & \dots & \dot{\psi}_N(t_f) \\ \vdots & \vdots & & \vdots \\ \psi_1^{(q)}(t_f) & \psi_2^{(q)}(t_f) & \dots & \psi_N^{(q)}(t_f) \end{bmatrix} \begin{bmatrix} \alpha_1^{[j]} \\ \alpha_2^{[j]} \\ \vdots \\ \alpha_N^{[j]} \end{bmatrix} = \begin{bmatrix} z_j(t_0) \\ \dot{z}_j(t_0) \\ \vdots \\ z_j^{(q)}(t_0) \\ z_j(t_f) \\ \dot{z}_j(t_f) \\ \vdots \\ z_j^{(q)}(t_f) \end{bmatrix}$$

- Or solve min-snap optimization

Example: planar quadrotor



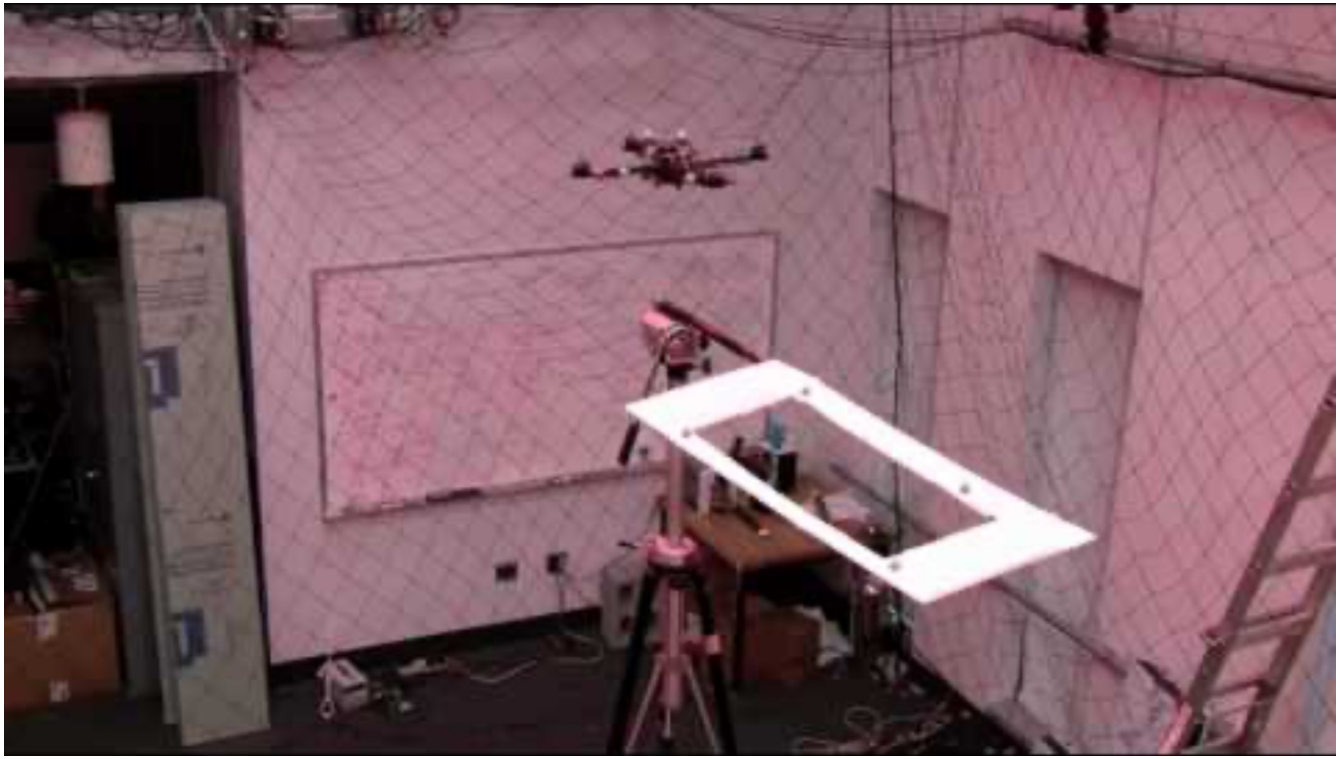
$$\begin{bmatrix} \dot{x} \\ \dot{v}_x \\ \dot{y} \\ \dot{v}_y \\ \dot{\phi} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} v_x \\ \frac{-(T_1+T_2)\sin\phi}{m} \\ v_y \\ \frac{(T_1+T_2)\cos\phi}{m} - g \\ \omega \\ \frac{(T_2-T_1)\ell}{I_{zz}} \end{bmatrix}$$

$$\mathbf{z} = \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\left.\begin{aligned} x &= x \\ v_x &= \dot{x} \\ y &= y \\ v_y &= \dot{y} \end{aligned} \qquad \begin{aligned} \phi &= \tan^{-1}\left(-\frac{\ddot{x}}{\ddot{y}+g}\right) \\ \omega &= \frac{\dddot{y}\ddot{x} - (\ddot{y}+g)\dddot{x}}{(\ddot{y}+g)^2 + \ddot{x}^2} \end{aligned}\right\} \beta$$

$$\mathbf{x} = \beta(\mathbf{z}, \dot{\mathbf{z}}, \ddot{\mathbf{z}}, \dddot{\mathbf{z}})$$

$$\mathbf{u} = \gamma(\mathbf{z}, \dot{\mathbf{z}}, \ddot{\mathbf{z}}, \dddot{\mathbf{z}}, \ddddot{\mathbf{z}})$$



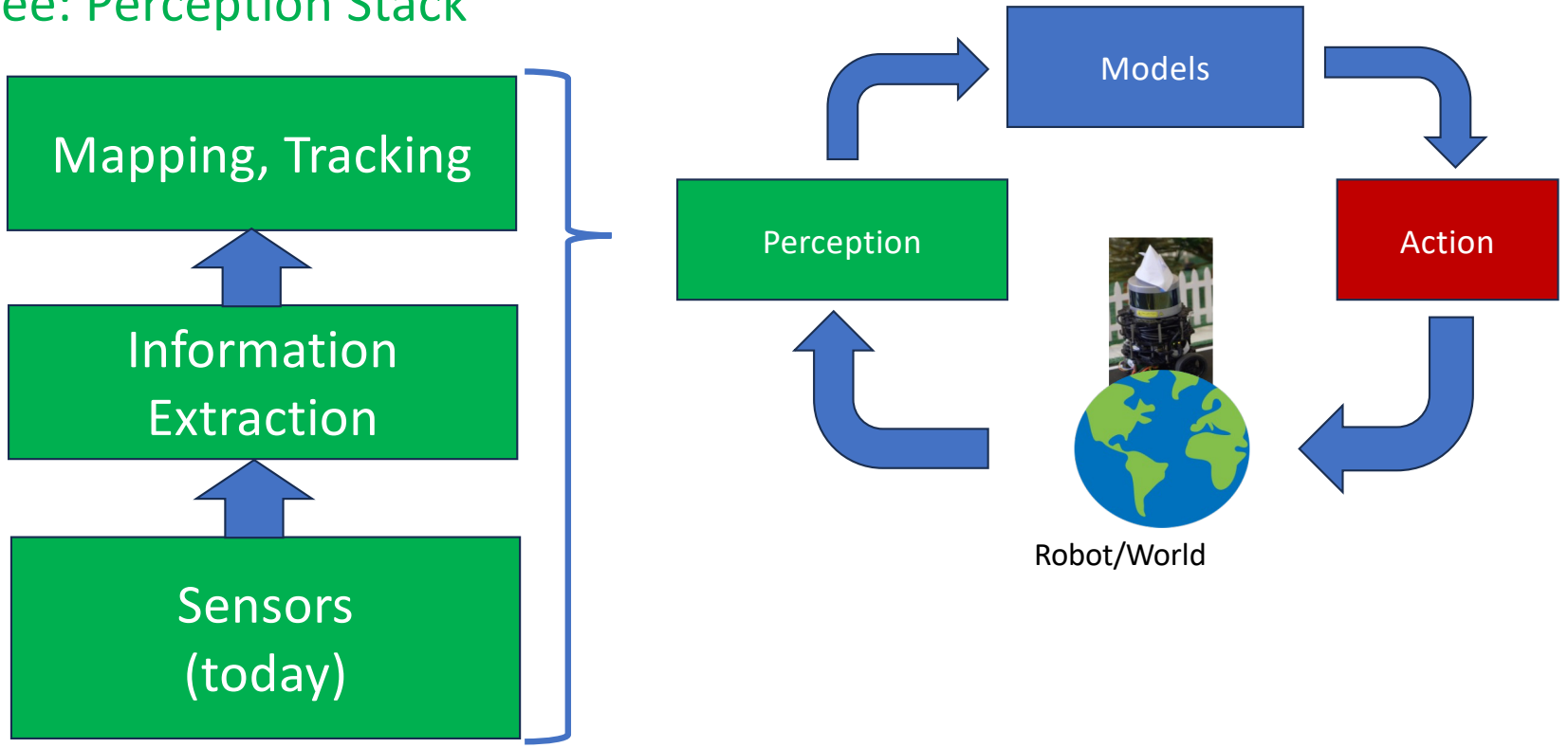
Mellinger, D., & Kumar, V., “Minimum snap trajectory generation and control for quadrotors,” In *2011 IEEE international conference on robotics and automation*, pp. 2520-2525, 2011.

When is a system differentially flat?

- The existence of a general, computable criterion to decide if the dynamical system $\dot{\mathbf{x}} = \boldsymbol{f}(\mathbf{x}, \mathbf{u})$ is differentially flat remains open
- Some results in this direction are, however, available
- Further readings:
  - Application to trajectory optimization:
    1. M. J. Van Nieuwstadt and R. M. Murray. Real-time trajectory generation for differentially flat systems. 1998
    2. R. M. Murray, M. Rathinam, and W. Sluis. Differential flatness of mechanical control systems: A catalog of prototype systems. 1995
    3. B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo. Robotics: modelling, planning and control. 2010
    4. D. Mellinger. Trajectory Generation and Control for Quadrotors. 2012.
  - Theory:
    1. J. Levine. Analysis and control of nonlinear systems: A flatness-based approach. 2009
    2. G. G. Rigatos, Gerasimos. Nonlinear control and filtering using differential flatness approaches: applications to electromechanical systems. 2015

Robot Perception

See: Perception Stack



Perception Stack: Computer vision, filtering, SLAM

Use sensor data to update the models

Localization, Mapping, Tracking:

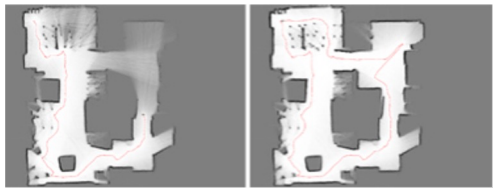
- EKF/Monte Carlo localization
- Occupancy grid mapping
- Pose graph optimization
- Tracking (EKF and Particle Filter)
- **AA273: Filtering (Schwager)**
- **AA275: Navigation (Gao)**

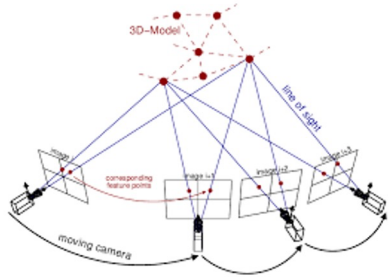
Information extraction

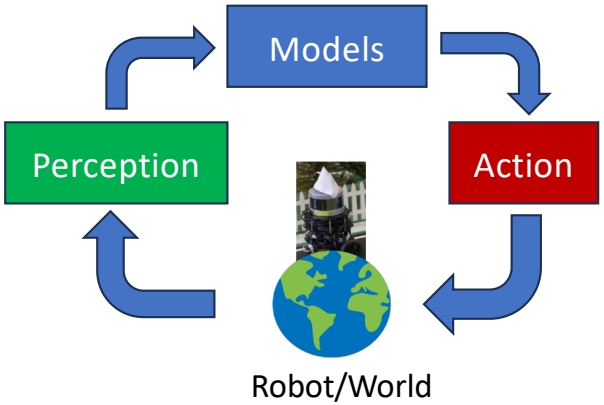
- Computer vision: features, correspondences, Structure from Motion (SfM), depth
- Lidar scan matching, ICP
- **CS231A: Comp Vision**

Sensors:

- RGB Cameras, RGB-D/stereo cameras, Lidar
- IMU, GPS, wheel encoders





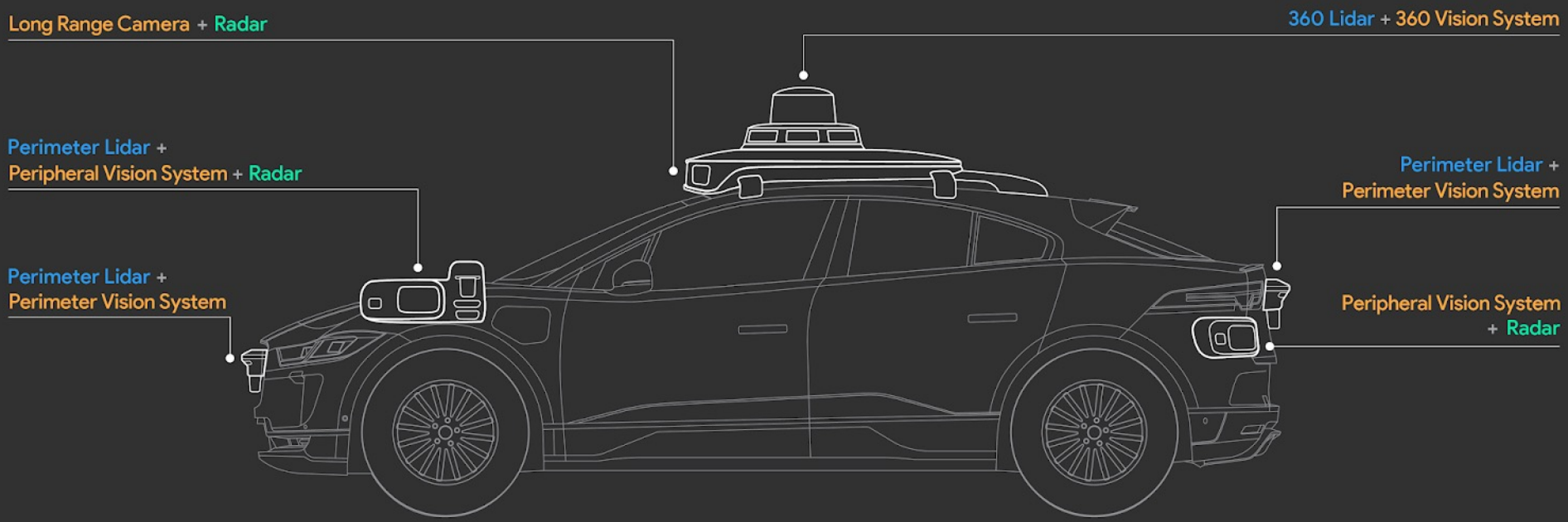


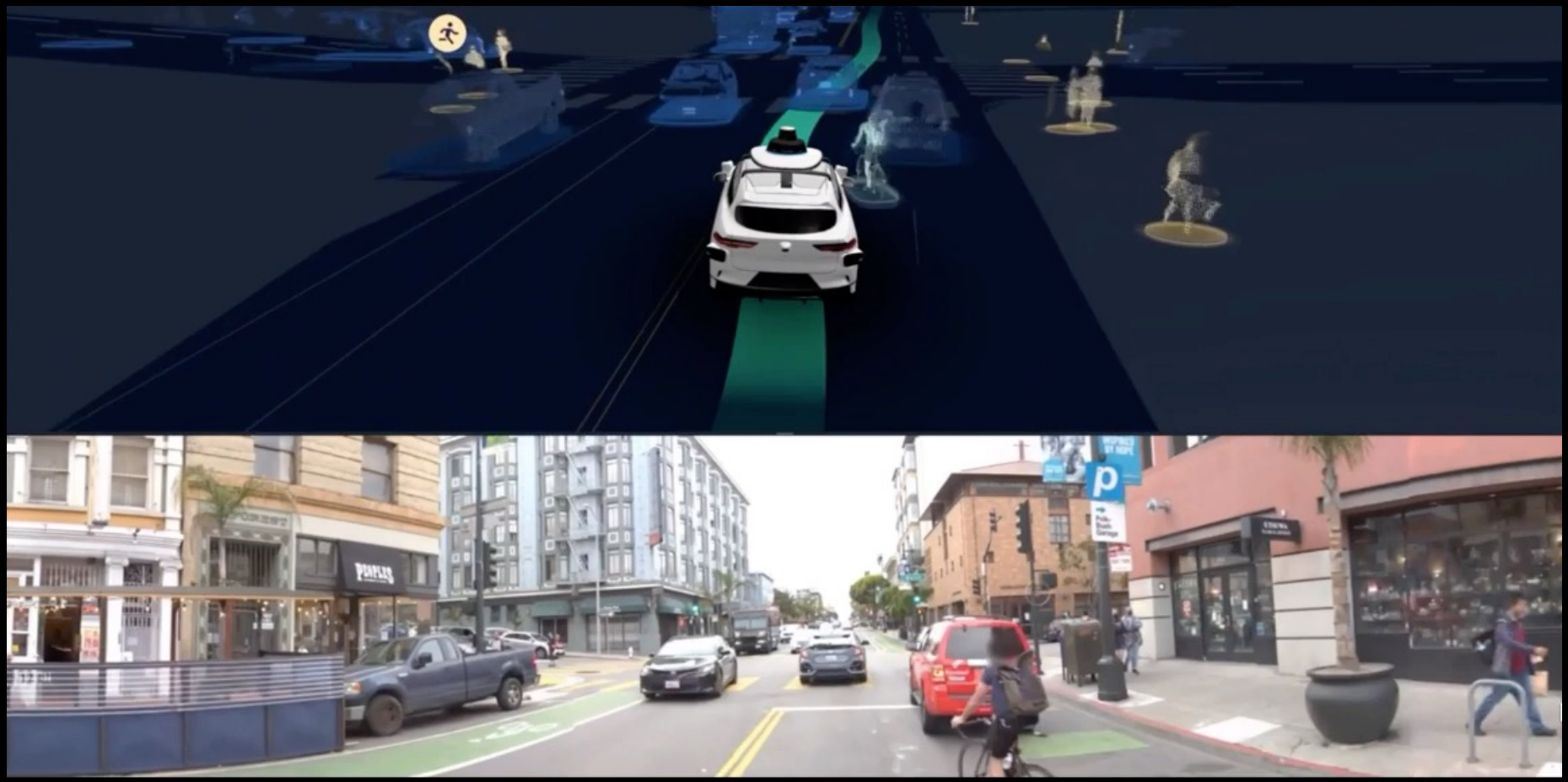
Sensors

Sensors

- Proprioceptive: IMU, joint and wheel encoders. GPS
- Exteroceptive: RGB, stereo, RGB-D, Lidar

Example: self-driving cars





Classification of sensors

- Proprioceptive: measure values internal to the robot
  - E.g.: robot acceleration, robot arm joint angles, and battery voltage
- Exteroceptive: acquire information from the robot's environment
  - E.g.: distance measurements and light intensity

- Passive: measure ambient environmental energy entering the sensor
  - Challenge: performance heavily depends on the environment
  - E.g.: temperature probes and cameras
- Active: emit energy into the environment and measure the reaction
  - Challenge: might affect the environment
  - E.g.: ultrasonic sensors and laser rangefinders

An ecosystem of sensors

- Joint and Wheel Encoders
- IMU
- GPS

Proprioceptive: non-navigational ego-states

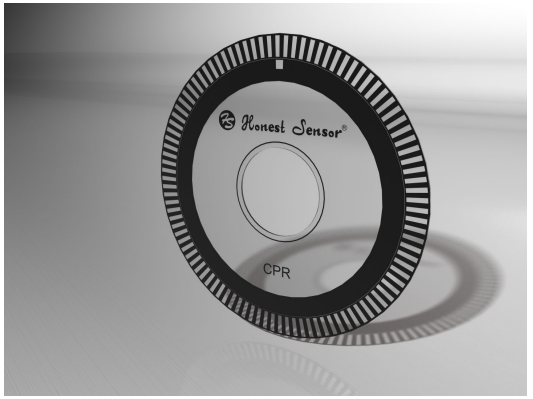
- RGB
- RGB-D
- Lidar
- Sonar
- Radar

Exteroceptive: relative poses of map features
color of map features
relative poses of targets
color of targets

Proprioceptive: Encoders

- Encoder: Measures angle of a wheel or joint.
- Mechanism: light emitter/receptor interrupted by black lines on a disk, resulting pulses indicate angle of rotation.
- Application:
  - Approximate robot localization
  - Called wheel odometry or “dead reckoning”

Wheel encoder
Credit: Pololu



Credit: Honest Sensor

Proprioceptive: Inertial Measurement Unit (IMU)

- Rate Gyroscope: measures angular velocity vector in body-fixed frame

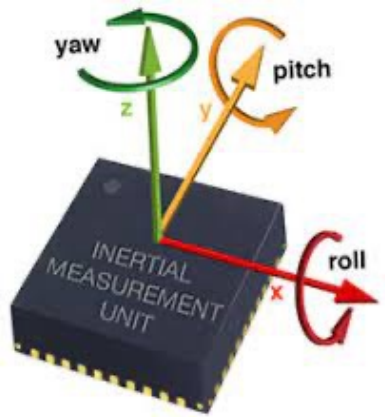
$$y_\omega = (\omega_x, \omega_y, \omega_z) + \nu_\omega$$ ← noise

- Accelerometer: measures linear acceleration vector in body-fixed frame (including gravity)

$$y_a = (a_x, a_y, a_z) + g + \nu_a$$

- Magnetometer: measures local magnetic field. Earth's + other magnetic disturbances. Compass.

$$y_B = (B_x, B_y, B_z) + \nu_B$$



IMU often pre-processed to get orientation

- Orientation: Low pass filter accel to get gravity vector (global down). Cross with magnetometer to get East. Cross East with down to get north.

$$y_a = (a_x, a_y, a_z) + g + \nu_a \qquad g \leftarrow \text{Filter}(y_a)$$

$$e_{\text{Down}} = \frac{g}{\|g\|} \qquad e_{\text{East}} = \frac{e_{\text{Down}} \times y_B}{\|e_{\text{Down}} \times y_B\|} \qquad e_{\text{North}} = \frac{e_{\text{East}} \times e_{\text{Down}}}{\|e_{\text{East}} \times e_{\text{Down}}\|}$$

- NED frame rotation matrix world-to-body:

$$R = [e_{\text{North}} \;\; e_{\text{East}} \;\; e_{\text{Down}}]$$

Proprioceptive: GPS (more generally, GNSS)

- Multi-lateration: use time-of-flight to get distance, known satellite locations to get end points. Then solve for receiver location with multi-lateration:

$$y_{\text{GPS}} = (q_x, q_y, q_z) + \nu_{\text{GPS}}$$

Exteroceptive: Active ranging

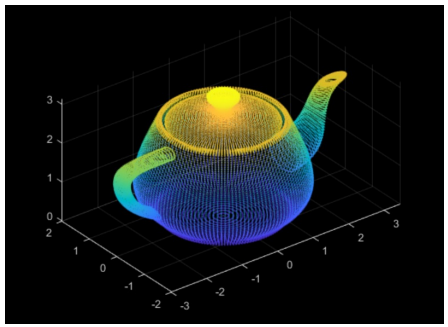
- Provide direct measurements of distance to objects in vicinity
- Key elements for both localization and environment reconstruction
- Main types:
  1. Time-of-flight active ranging sensors (e.g., ultrasonic and laser rangefinder)

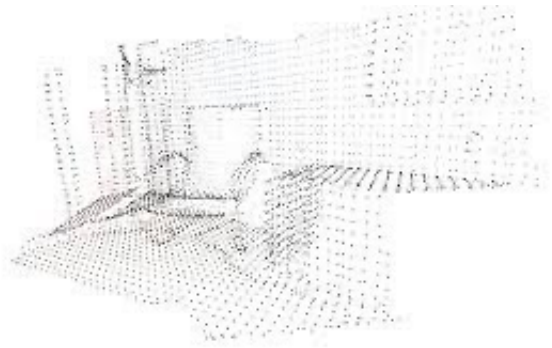
Credit: https://electrosome.com/hc-sr04-ultrasonic-sensor-pic/

  2. Geometric active ranging sensors (optical triangulation and structured light)

Point clouds

- All these sensors output point cloud data

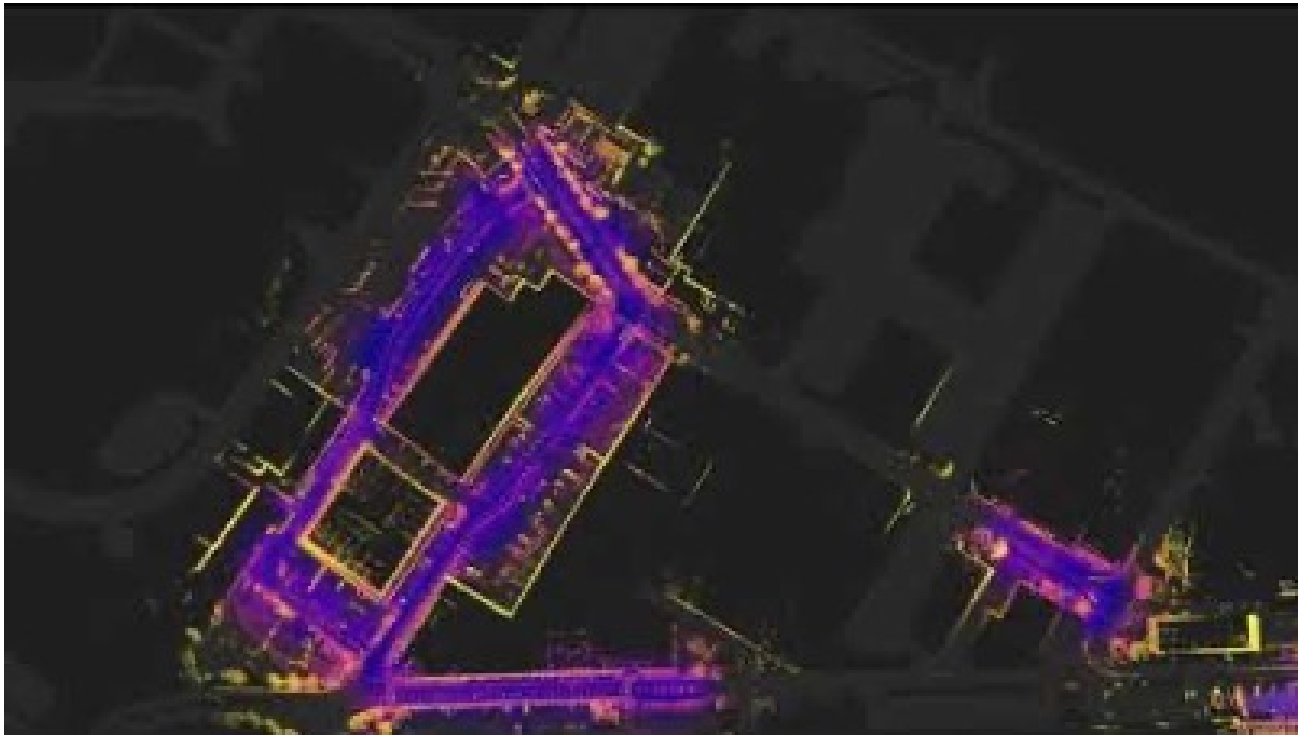




$$P = (p_1, p_2, \ldots, p_N), \quad p_i = (p_x, p_y, p_z)^T$$

Example Lidar data from nuScenes dataset

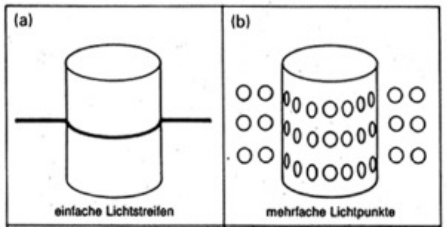
https://www.nuscenes.org/



Lidar point clouds overlayed with Camera images https://www.nuscenes.org/

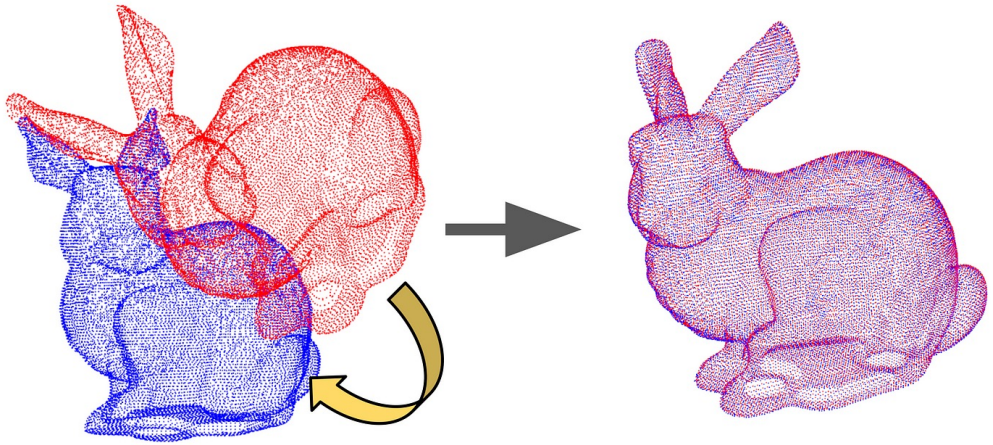
Geometric active ranging

- Fundamental principle: use geometric properties in the measurements to establish distance readings
- The sensor projects a known light pattern (e.g., point, line, or texture); the reflection is captured by a receiver and, together with known geometric values, range is estimated via triangulation
- Examples:
  - Optical triangulation (1D sensor)
  - Structured light (2D and 3D sensor)



Credit: Matt Fisher

Classic Problem: Point Cloud Alignment



- Point cloud alignment: Align point cloud A (from a new scan) to point cloud B (from a previous scan or map)
- Obtain the relative pose: The pose transform required to move P_A to align with P_B

(τ_{BA}, R_{BA})

Translation vector

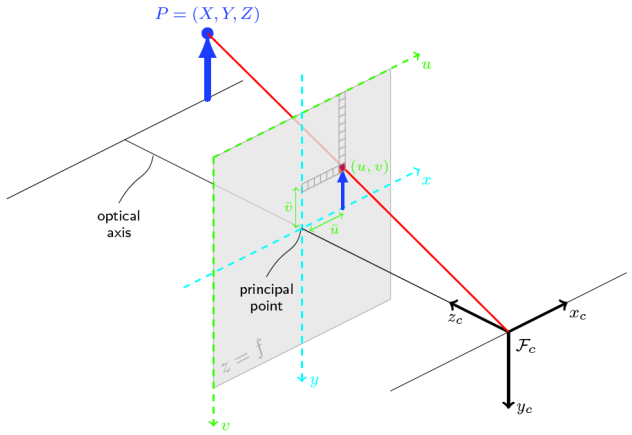
Rotation matrix

RGB (monocular vision)

- Produces a matrix of RGB color vectors

$$I = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1N} \\ c_{21} & c_{22} & \cdots & c_{2N} \\ \vdots & \ddots & & \vdots \\ c_{M1} & \cdots & & c_{MN} \end{bmatrix}$$

$$c_{uv} = (r_{uv}, g_{uv}, b_{uv})$$



- Fundamental problem: Where in the world frame was the "thing" that produced pixel c_{uv}. How to go from (u, v) pixel coordinates to (p_x, p_y, p_z) world coordinates?

Calibration matrix:

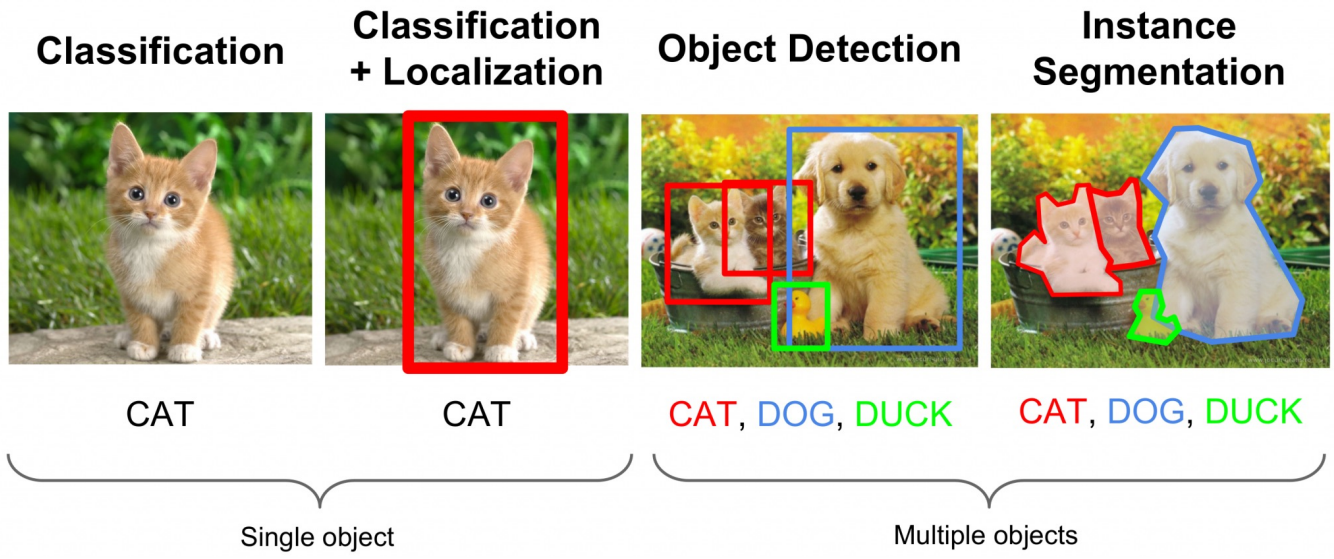
$$K = \begin{bmatrix} \alpha & 0 & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

Pinhole camera model:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}$$

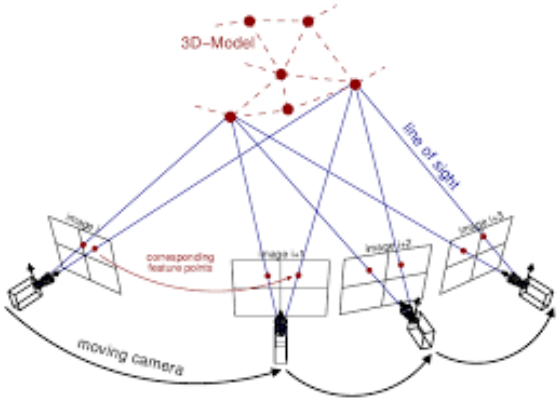
Classic Vision Problems: Semantic

- Classification: What's in the image?
- Detection: Draw bounding boxes around objects with labels
- Segmentation: Identify pixels (masks) belonging to object classes

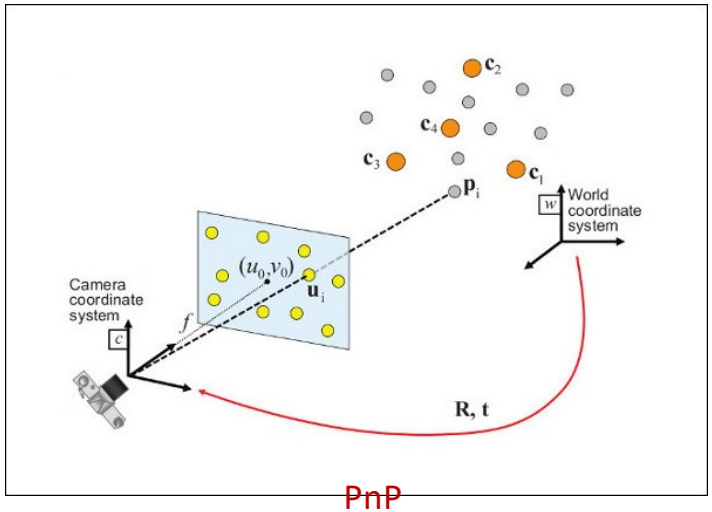


Classic Vision Problems: Geometric

- Structure from Motion (SfM, multi-view geometry): Find 3D locations of points in scene based on pairs or collections of images from different perspectives. Gives a point cloud and camera poses.
- N Point Perspective (PnP): Match 2D points in the image with 3D points in a pre-existing 3D model



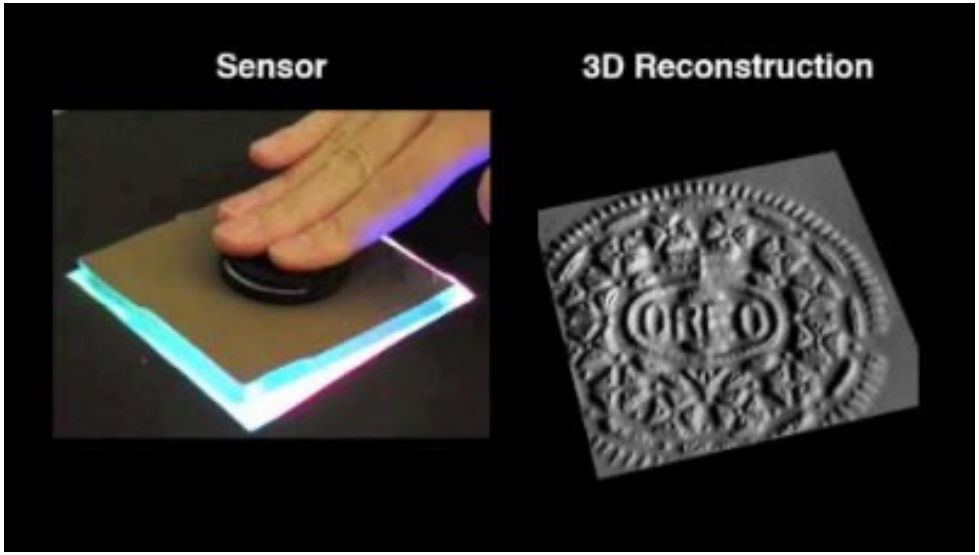
SfM



PnP

Several other sensors are available

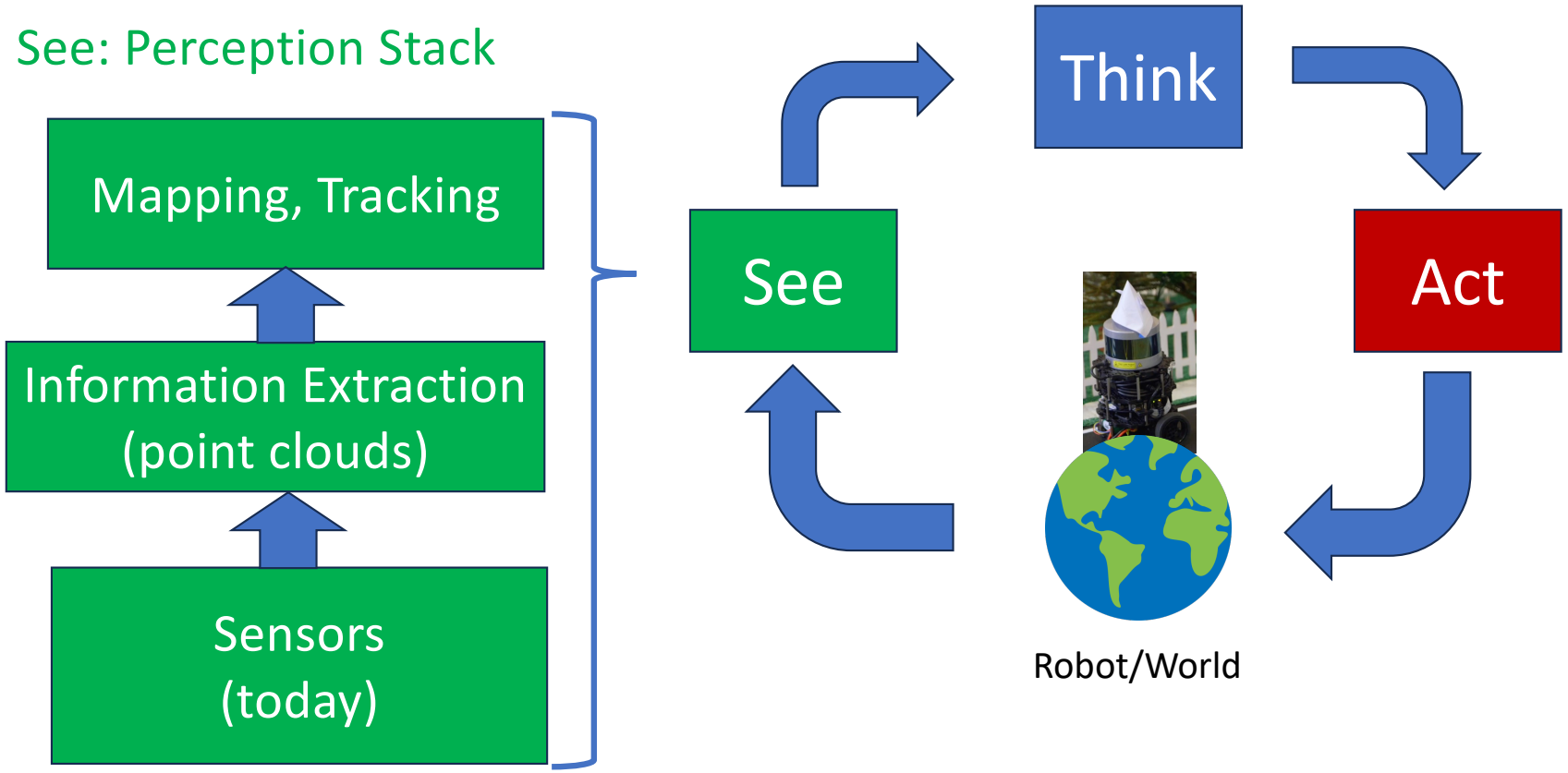
- Classical, e.g.:
  - Radar (possibly using Doppler effect to produce velocity data)
  - Tactile sensors, load cells
- Emerging technologies:
  - Artificial skins
  - Neuromorphic cameras



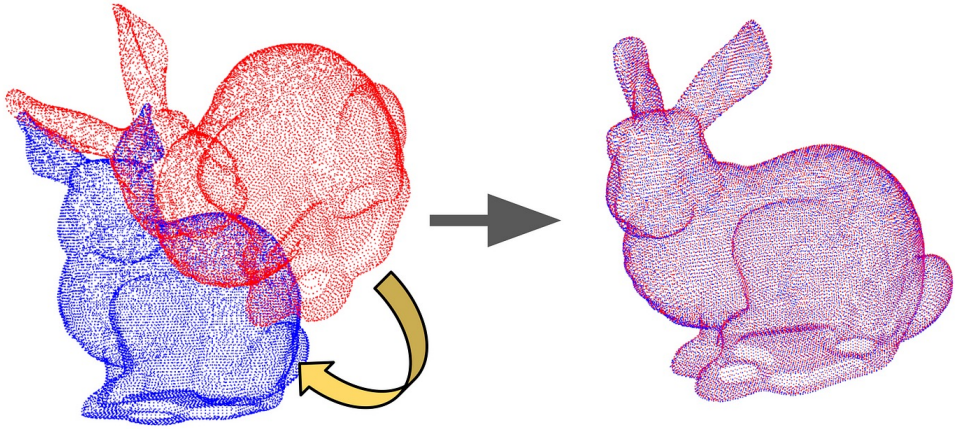
GelSight at Emerging Technologies at SIGGRAPH 2009
https://www.gelsight.com/

Robot Perception

See: Perception Stack



Point cloud alignment: Iterative Closest Point (ICP)



- Obtain the relative pose: The pose transform required to move P_A to align with P_B

$$(\tau_{BA}, R_{BA})$$

Translation vector

Rotation matrix

Basics and notation:

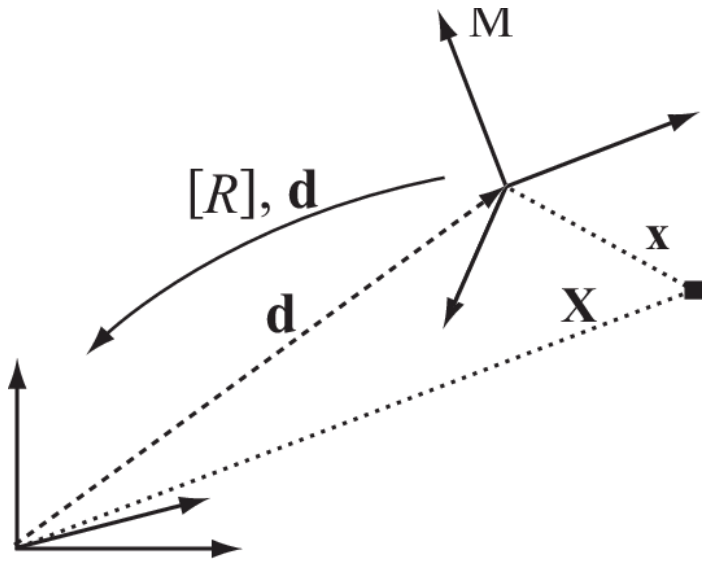
- Rigid body transforms, SE(2), SE(3)

$$(\tau_{BA}, R_{BA}) \in \mathrm{SE}(2) \quad \text{or} \quad \mathrm{SE}(3)$$

$$P_B = R_{AB} P_A + \tau_{BA}$$

rotate

translate



Alignment with Known Correspondences

- Find (τ_{BA}, R_{BA})

$$P_B = R_{AB} P_A + \tau_{BA}$$

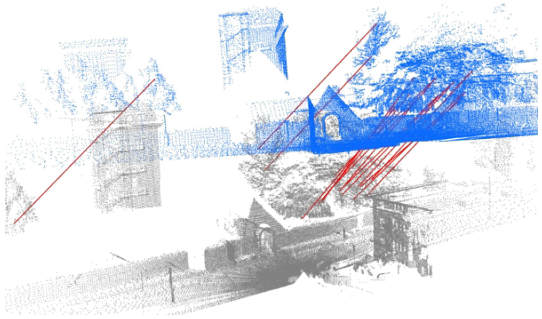
$$\min_{(\tau_{BA}, R_{BA})} \|P_B - (R_{BA} P_A + \tau_{BA})\|_{\mathcal{F}}^2$$
$$\text{s.t.} \quad R_{BA}^T R_{BA} = I$$

- Known solution based on SVD

ICP: iterate with closest point heuristic

- For each point in P_A

  p_i^A match to closest point p_j^B from P_B

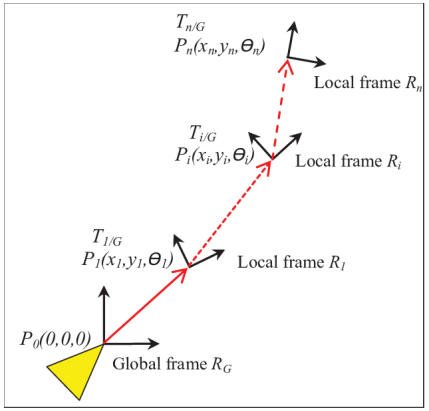


- Solve for (τ_{BA}, R_{BA})
- Repeat until convergence

Why?

- Lidar odometry: Gives relative poses for robot between successive Lidar scans

- SLAM: Lidar “front end” for pose graph optimization (PGO) “back end,” to find global map and robot’s trajectory (and current pose) in the map



Next time: Computer vision