Simultaneous Localisation and Mapping

COSC450

Simultaneous Localisation and Mapping

Localisation:

- Input: A 3D model and 2D image
- Output: Pose of the camera

Challenges:

- 3D-2D feature matching
- Robust pose estimation
- Dealing with ambiguity

Mapping:

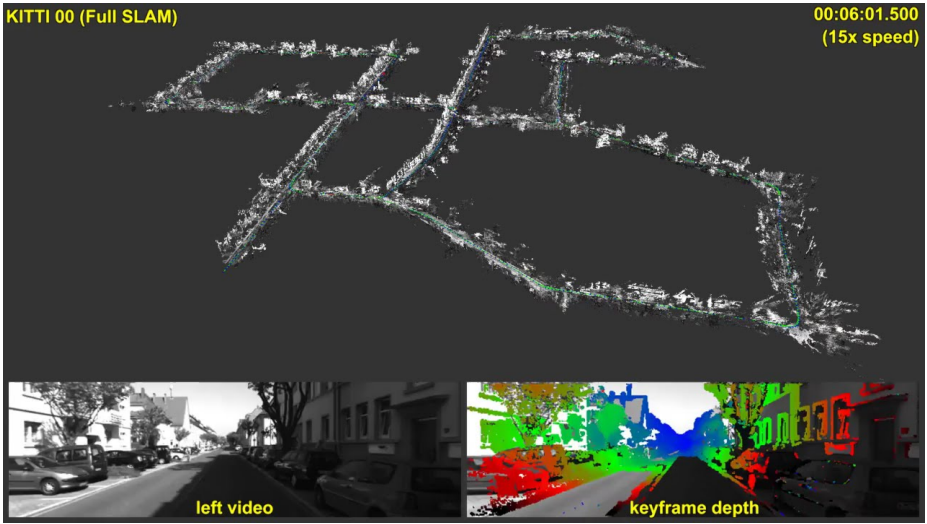
- Input: 2D images (and camera poses?)
- Output: A 3D model of the world

Challenges:

- Determining camera pose
- Dealing with large areas
- Drift / loop closure

Doing two hard things at once – is it twice as hard?

SLAM Example – Large Scale Direct SLAM



https://www.youtube.com/watch?v=oJt3Ln8H03s, Engel, Stückler & Cremers, IROS 2015

SLAM Problem Formulation

- We have a sequence of observations, $\mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_t$
- We want to estimate a map, $\mathbf{m}_1, \mathbf{m}_2, \ldots \mathbf{m}_t$, and location, $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t$, over time
- This is often represented in a probabilistic framework:

$$P(\mathbf{m}_t, \mathbf{x}_t | \mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_n)$$

- Expectation-Maximisation algorithms split this into two parts:
  1. Assuming we know the map, what is our pose? Estimate $P(\mathbf{x}_t | \mathbf{m}_t, \mathbf{x}_{t-1}, \mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_t)$
  2. Assuming we know the pose, what is the map? Estimate $P(\mathbf{m}_t | \mathbf{x}_1, \mathbf{m}_{t-1}, \mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_t)$

  What do we expect our pose to be? Maximise the map probability based on this pose.
- Often we can just update based on the latest measurements:

$$P(\mathbf{x}_t | \mathbf{m}_t, \mathbf{x}_{t-1}, \mathbf{o}_t[, \mathbf{o}_{t-1}]) \qquad P(\mathbf{m}_t | \mathbf{x}_t, \mathbf{m}_{t-1}, \mathbf{o}_t[, \mathbf{o}_{t-1}])$$

FastSLAM

Fast Slam (Montemerlo *et al.*, 2002) uses particle and Kalman filters

- Simple case – assume correpsonding 2D features between frames as observations
- The map consists of K 3D landmark points, $m_1, m_2, \ldots, m_k$
- Assuming the landmarks' measurements are independent lets us factor the problem as

$$P(\mathbf{m}_t, \mathbf{x}_t | \mathbf{o}_t) = P(\mathbf{x}_t | \mathbf{o}_t) \prod_{j=1}^{k} P(m_j | \mathbf{x}_t, \mathbf{o}_t)$$

- A particle filter is used to predict the new pose
- A Kalman filter is then used to estimate the map probability

FastSLAM – Particle Filter

We want to estimate $P(\mathbf{x}_t|\mathbf{o}_t)$

- We use a particle filter from $\mathbf{x}_{t-1}$
- This gives us an *a priori* estimate of $P(\mathbf{x}_t)$
- We need to update this based on the measurements, $\mathbf{o}_t$
- To do this we need to predict the observations...
- ...and this depends on the map

FastSLAM – Extended Kalman Filter

The map is first estimated for *each particle*

- We want to estimate $P(m_j|\mathbf{x}_t, \mathbf{o}_t)$
- We know (recursively) the probabilities from the previous frame
- We have a model of how the measurements depend on state and map
- This is generally non-linear – an *extended* Kalman filter
- One Kalman filter per 3D measurement (m_j) per particle
- Lots of EFKs, but all just 2D (measurement)
- This gives a lot of estimates for each 3D point (1 per particle)
- All Gaussians, so easy to combine, so we have a map estimate

Other SLAM Approaches

PTAM (Parallel Tracking and Mapping)

- AR needs real-time tracking
- Fast map update less important
- Separate tracking & mapping threads
- Tracking runs in real-time
- Pyramid-based matching of FAST features with affine transforms
- Dense mapping from keyframes
- Bundle adjustment to minimise reprojection error

ORB-SLAM

- ORB features used throughout
- Sparse map from features
- Large-scale mapping possible
- Only local updates each frame
- Relocalisation after tracking failures
- Graph-based reduction of camera pairs
- Loop closure

Other SLAM Approaches

LSD-SLAM (Large-Scale Dense SLAM)

- Whole images, not features
- Depth estimated by stereo methods
- Keyframes used as reference for tracking
- Gives very dense maps

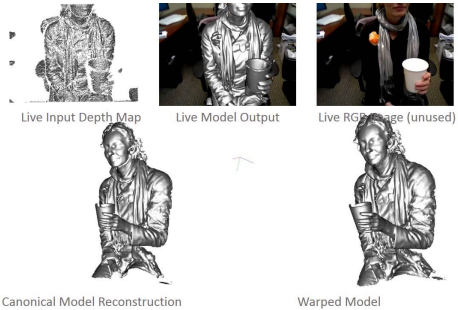
SLAM++

- Objects detected with CNN
- These are the features
- Leads to a compact map with semantic meaning attached

KinectFusion

- Uses depth camera (Kinect)
- Voxel-based map
- Voxels store distance to surface

DynamicFusion



https://www.youtube.com/watch?v=oJt3Ln8H03s

Drift and Loop Closure

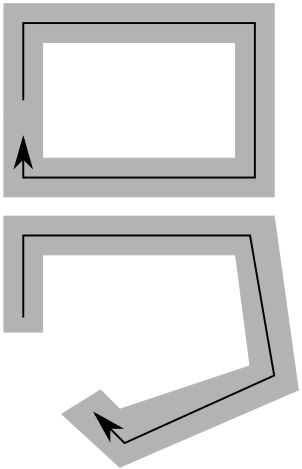
SLAM updates map/location frame-by-frame

- There are small errors each frame
- These accumulate over time
- Eventually the map becomes inaccurate

Loop closure provides corrections

- Detect when you return to a location
- Can then determine the error...
- ...and fix your path around the loop

How to detect when you return to a place?



Bag-of-Words in Document Search

Inspiration from information retrieval

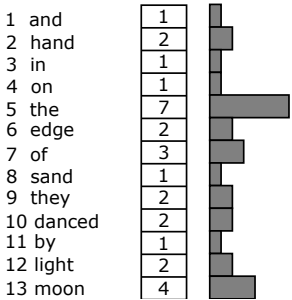
- Document classification and search
- Analyse documents for common words

Bag-of-Words (BoW) models are common

- Each word is given an index
- Documents represented as an array
- Array records frequency of each word
- Related words are grouped together
  compute computer computing

And hand in hand on the edge of the sand
They danced by the light of the moon,
the moon,
the moon,
They danced by the light of the moon.

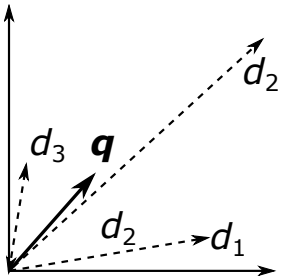
Edward Lear, *The Owl and the Pussycat*



Comparing Bags-of-Words

Distance between histograms

- Can view BoW as a vector
- Euclidean distance?
  - What happens if you compare a document to itself repeated twice?
- Can normalise lengths
- Can just have binary vectors
  - Presence/absence of words, not count
- Can use angle between vectors
  - Easier to compute the cosine



$$\cos\theta(\mathbf{q}, \mathbf{d}_i) = \frac{\mathbf{q} \cdot \mathbf{d}_i}{\|\mathbf{q}\|\|\mathbf{d}_i\|}$$

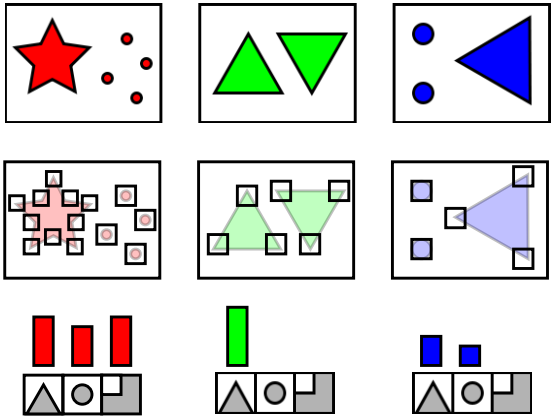
Visual Bag-of-Words

How does this relate to images?

- Documents naturally divide into words
- Feature descriptors can be used
- But features are rarely identical
- So cluster them to make a dictionary

Visual Bag-of-Words

- Given a large collection of descriptors
- Apply k-means for some large k
- This gives a vocabulary of k words
- Images can now be described by a BoW



Using Visual Bags-of-Words

Image Search

- Find similar images to a query image
- Compute BoW for each image in the database that we want to search
- Organise these into a *k*d-Tree
- Given a query, compute its BoW
- Return approximate nearest neighbours using the *k*d-Tree

Object Classification

- We have a set of labelled images
- We want to label a new image
- Compute BoW for each labelled image
- Find average BoW for each label
- Compute query image BoW
- Assign it the nearest label