Visual Servo Control

Roel Pieters (r.s.pieters@tue.nl)

Embedded Vision in Dynamics and Control
Mechanical Engineering
Eindhoven University of Technology
The Netherlands

Technische Universiteit
Eindhoven
University of Technology

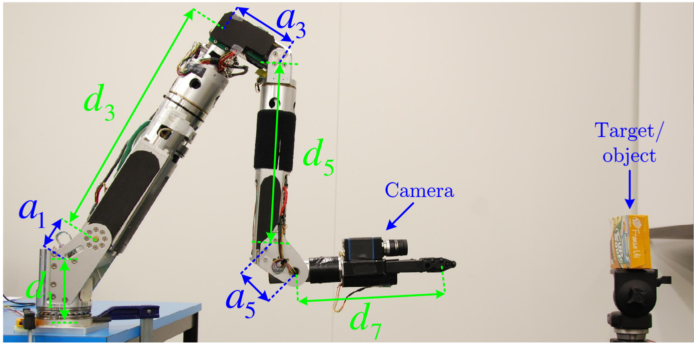
March 2012

Contents

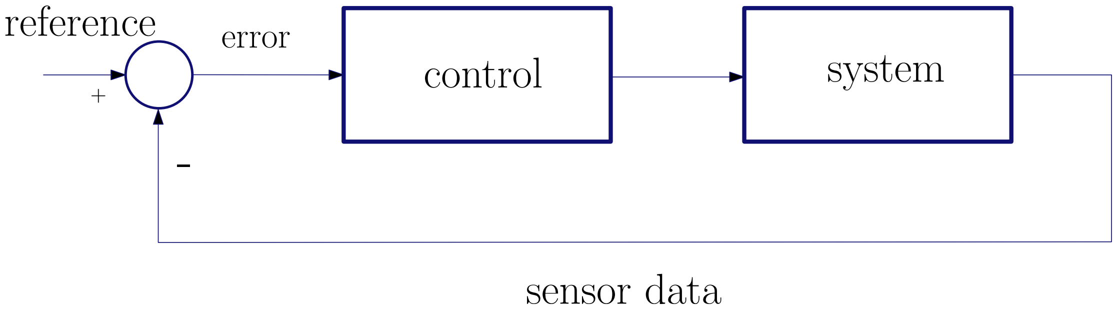
1. Visual Servo Control

Controlling Robots using visual information

- Camera location: Eye-in-hand vs. Eye-to-hand
- Camera: mono vs. stereo (omnidirectional)
- Control: image-based vs. position-based
- Processing: remote (pc) vs. local (embedded)

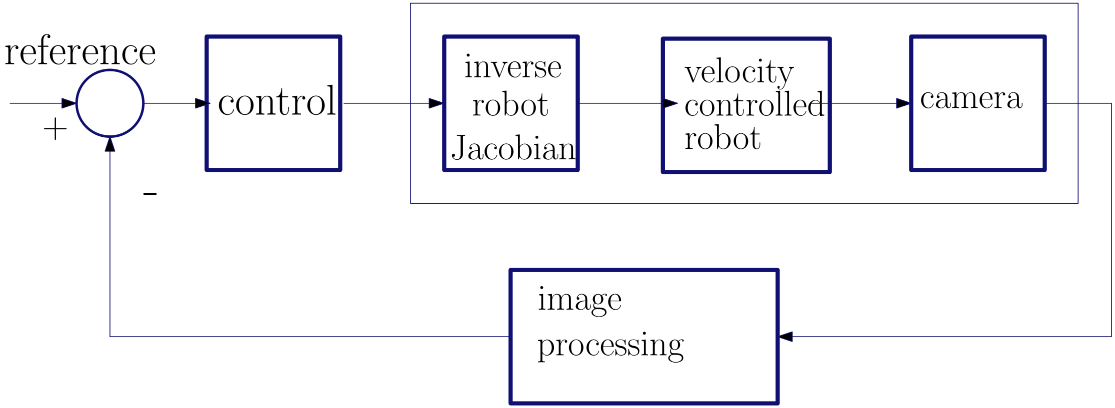


Feedback Control



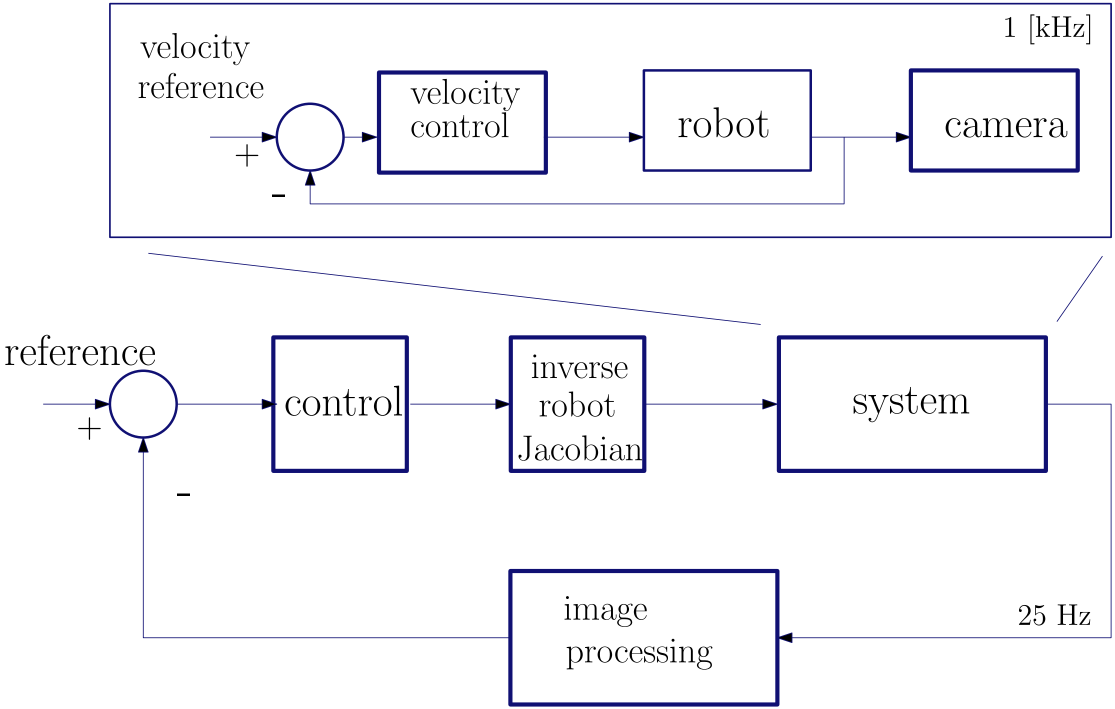
- System: Robot, quadcoptor, car
- Control: PID, etc
- Reference: position, velocity, torque
  Robot: use encoders
  Quadcoptor: use gyroscope, accelerometer

Vision-based Control



- System: velocity controlled robot + camera
- Control: ?
- Reference:
  outer loop: image data (?)
  inner loop velocity

Vision-based Control



- Velocity controlled robot: PI control
- Vision loop for goal/target

Visual Servoing

Image Based Visual Servoing

- Control in image space
- Image processing

Position Based Visual Servoing

- Control in Cartesian space
- Image processing + Pose estimation

Visual Servoing

Image Based

- eye-in-hand system
- error: $\mathbf{e}(t) = \mathbf{s} - \mathbf{s}^*$ ($\mathbf{s}$ = feature vector)
- find: relation between camera velocity $\mathbf{v}_c$ and feature velocities $\dot{\mathbf{s}}$

$$\dot{\mathbf{s}} = \mathbf{L}_s \mathbf{v}_c \tag{1}$$

- $\mathbf{L}_s$: image Jacobian, interaction matrix. $\mathbf{s}^*$: constant

Error velocity:

$$\dot{\mathbf{e}}(t) = \mathbf{L}_e \mathbf{v}_c \tag{2}$$

where $\mathbf{L}_e = \mathbf{L}_s$. Design control of $\mathbf{v}_c$ as exponential decay: $\dot{\mathbf{e}} = -\lambda \mathbf{e}$

$$\mathbf{v}_c = -\lambda \mathbf{L}_e^+ \mathbf{e} \quad \rightarrow \quad \dot{\mathbf{e}} = -\lambda \mathbf{L}_e \mathbf{L}_e^+ \mathbf{e} \tag{3}$$

Image Based

How to get Image Jacobian?
With pinhole camera model, relate image feature velocities to camera velocity

- 1 image feature $\mathbf{s} = [u, v]^T$: 2D, so we get 2×6 image Jacobian
- find: relation between camera velocity $\mathbf{v}_c$ and feature velocities $\dot{\mathbf{s}}$
- $\mathbf{v}_c = [t_x, t_y, t_z, \omega_x, \omega_y, \omega_z]^T$ = camera velocity

$$\dot{\mathbf{s}} = \mathbf{L}_s \mathbf{v}_c, \tag{4}$$

where the image Jacobian is defined as

$$\mathbf{L}_s = \begin{bmatrix} \frac{1}{z} & 0 & -\frac{u}{z} & -uv & 1+u^2 & -v \\ 0 & \frac{1}{z} & -\frac{v}{z} & -(1+v^2) & uv & u \end{bmatrix} \tag{5}$$

$\mathbf{L}_s$ is underdetermined, cannot be inverted: use more points! (>3)

Visual Servoing

Stability

Prove stability with a Lyapunov function $\mathcal{L}$, which can be seen as energy of the system

Thus, if $\mathcal{L} > 0$ and $\dot{\mathcal{L}} < 0$, we have stability

$$\mathcal{L} = \frac{1}{2}\mathbf{e}^T\mathbf{e} \tag{6}$$

$$\dot{\mathcal{L}} = \mathbf{e}^T\dot{\mathbf{e}}, \quad \dot{\mathbf{e}} = \mathbf{L}_e\mathbf{v}_c \tag{7}$$

$$\mathbf{v}_c = -\lambda\mathbf{L}_e^+\mathbf{e} \tag{8}$$

$$\dot{\mathcal{L}} = -\lambda\mathbf{e}^T\mathbf{L}_e\mathbf{L}_e^+\mathbf{e} \tag{9}$$

- $\mathbf{L}_e\mathbf{L}_e^+ > 0$ for stability
- Actually $\mathbf{L}_e^+$ is estimated, so only local asymptotic stability
- what about camera calibration?

Visual Servoing

Position Based

Pose error: $\mathbf{e} = (\mathbf{t_h}, \theta\mathbf{u})^{\mathrm{T}}$
Again: $\dot{\mathrm{e}}(t) = \mathbf{L}_e \mathbf{v}_c$
Interaction matrix now becomes:

$$\mathbf{L_e} = \begin{bmatrix} -\mathbf{I}_3 & [\mathbf{t}_h]_\times \\ 0 & \mathbf{L}_{\theta\mathbf{u}} \end{bmatrix}, \theta = \mathrm{acos}\left(\frac{\mathrm{trace}(\mathbf{R}) - 1}{2}\right), \mathbf{u} = \frac{1}{2\sin(\theta)} \begin{bmatrix} R_{32} - R_{23} \\ R_{13} - R_{31} \\ R_{21} - R_{12} \end{bmatrix}.$$

$$\mathbf{L}_{\theta\mathbf{u}} = \mathbf{I}_3 - \frac{\theta}{2}[\mathbf{u}]_\times + \left(1 - \frac{\mathrm{sinc}(\theta)}{\mathrm{sinc}^2(\frac{\theta}{2})}\right)[\mathbf{u}]_\times^2, \quad (10)$$

This leads to:

$$\mathbf{v}_c = \begin{bmatrix} \boldsymbol{v}_c \\ \boldsymbol{\omega}_c \end{bmatrix} = \begin{bmatrix} -\lambda(\mathbf{t}_h + [\mathbf{t}_h]_\times \theta\mathbf{u}) \\ -\lambda\theta\mathbf{u} \end{bmatrix}. \quad (11)$$

Visual Servoing

Position Based

- Simply said; $\mathbf{v}_c$ contains pose error
- Stability proof is similar.
  If pose estimation is perfect, then global asymptotic stability
- Estimation and camera calibration play a big role
- Noise is a problem for estimation
- Note that PBVS is very similar to traditional robot control
  (PBVS simply assumes a pose error is available)
- How to get this pose error?
  - image processing from CAD data
  - image processing planar estimation
  - etc..

Visual Servoing

(Dis)advantages

IBVS

- + image space motion
- + robust to noise and errors
- + 3D model not necessary
- - can give singularities
- - motion of Cartesian space unknown

PBVS

- + Cartesian space motion
- - sensitive to noise and errors
- - 3D model necessary

Hybrid/switched approaches and more

Combination of IBVS and PBVS

- IBVS for translation
- PBVS for rotation

switching based on some performance criteria

- IBVS when feature about to leave the FOV
- PBVS otherwise

Addition of both

- PBVS to guarantee global stability and performance
- IBVS to maintain FOV constraint

Questions

Keypoint detection

2. Keypoint detection

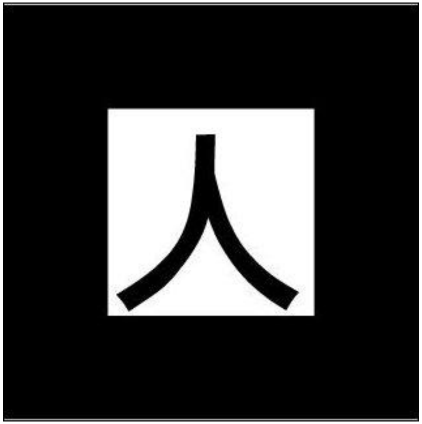
Motion transformation between two frames can be determined from two image views.

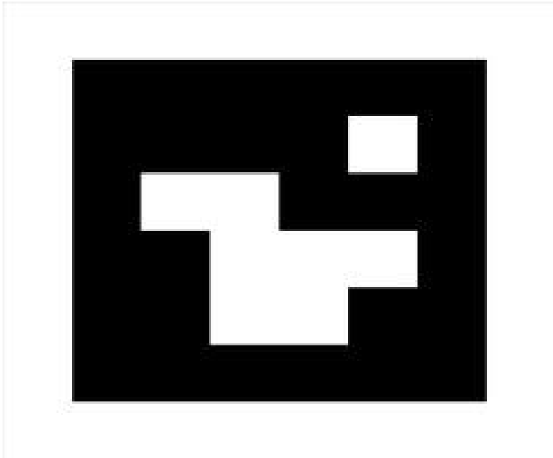
The idea is that an object as used for reference, can be encoded and reduced to a number of 'interesting' points. The transformation between two views can then be found from the two point sets.

These points can be simple (markers) or complex (natural features).

Keypoint detection

Marker-based





- Augmented Reality (ARToolkit)
- easy, fast and robust image processing
- clutters space

Keypoint detection

Natural features

Use points of objects themselves as reference points

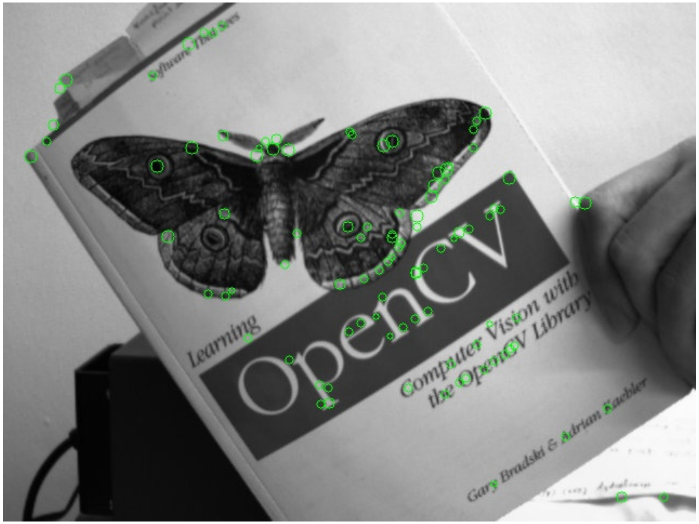
- each selected point from reference has to be found with respect to changes (e.g. transformation, size, intensity)
- much more computationally demanding
- Many keypoint detectors exist (SIFT, SURF, ORB, etc)
- depending on conditions and available resources choose a detector
- for instance offline vs. online, affine transformation

Keypoint detection

Natural keypoint detection

Properties of ideal keypoint

- Local and invariant
- Robust to: noise, blur, discretization, compression
- Distinctive
- Efficient vs. accurate



Keypoint detection

SURF

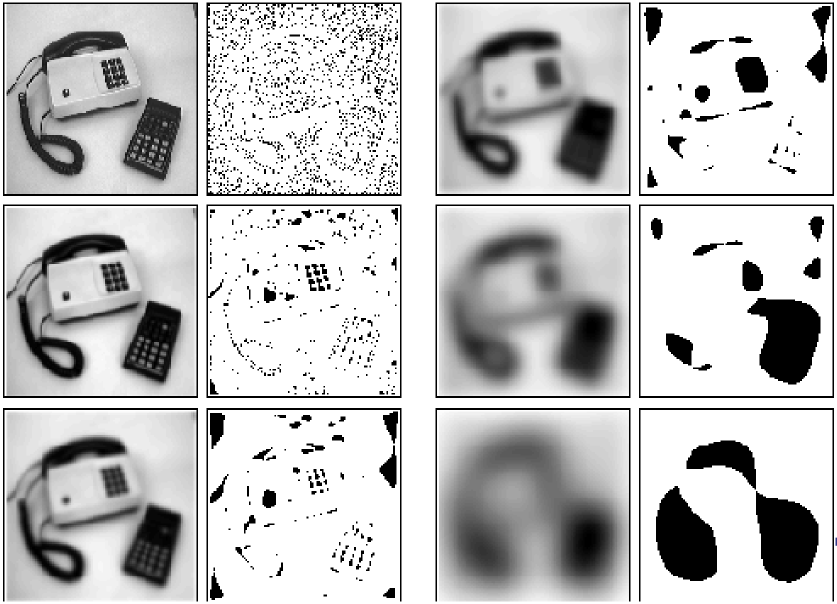
Speeded Up Robust Features

- Simplification of SIFT (Scale Invariant Feature Transform)
- Adapted to be as computationally efficient as possible
- Detector + Descriptor
  - Detect points based on 'uniqueness' against several factors (in scale-space)
  - Descript each point by a vector to have its own 'identity'

Keypoint detection

Scale-space

- Select points in scale-space: space of scaled down images
- Why? Same points should be found for whole depth range
- How? image(blurred) - image(less-blurred)
  Blurring: Convolute image with Gaussian kernels



Keypoint detection

Detector

- Scale-space: Difference of Gaussian (implemented as Determinant of Hessian $\rightarrow 2^{nd}$ order partial derivatives)
- Scale-space is divided in octaves (new octave corresponds to the doubling of the kernel size)
- Select keypoints with non-maxima suppression ($3 \times 3 \times 3$ window) (also in scale space)
- Local orientation of each keypoint in scale-space is computed from the local neighbourhood around each keypoint
  most weight in certain direction

- we now have points in different scales at different locations with an orientation

Keypoint detection

Descriptor

- Feature vector (dimension 64 or 128) built upon:

- keypoint's orientation and gradient:
- 16 subregions around keypoint
- each subregion computes (gradients):
  - Sum of dx
  - Sum of dy
  - Sum of abs(dx)
  - Sum of abs(dy)

- this vector now represents ONE keypoint

Keypoint detection

Matching

Remember: we have 2 sets of keypoints and descriptors

- Again, many possibilities for matching:
- Nearest neighbours
- Brute force search
- RANSAC

Result after matching: a set of corresponding points
$\longrightarrow$ input for homography estimation

Keypoint detection

DEMO!

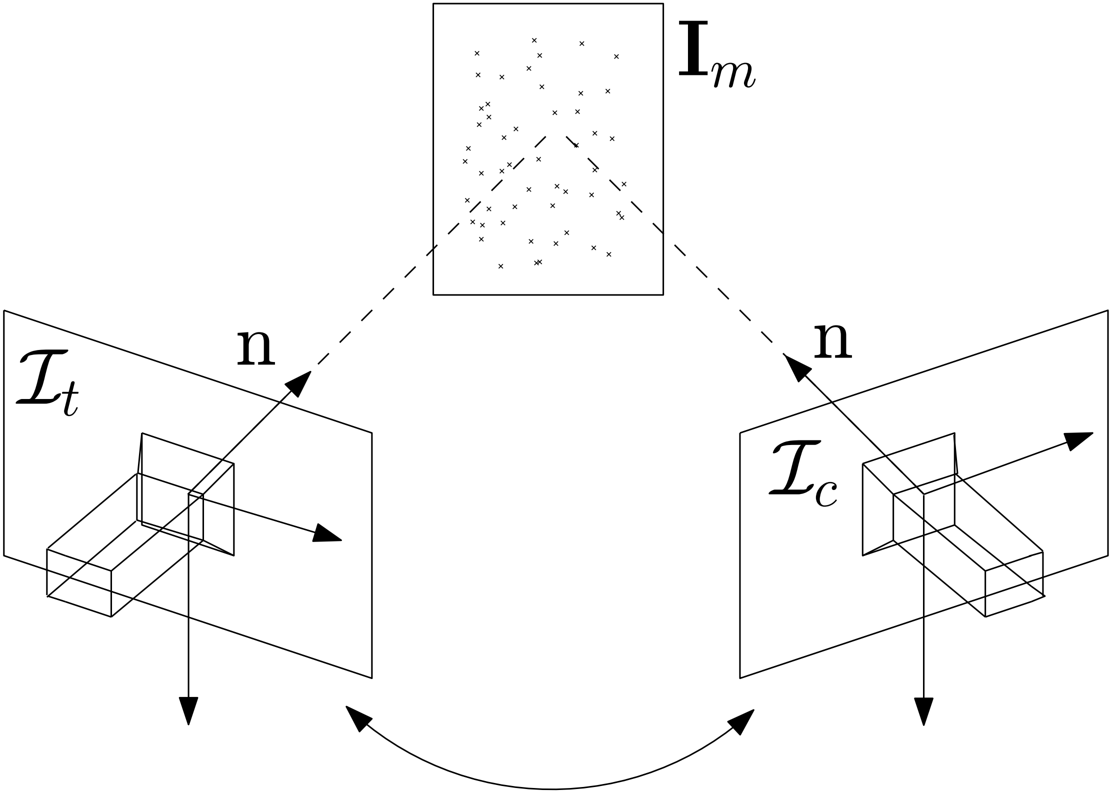
3. Homography

Geometric Visual Transformations:

- Perspective projection
  Mapping a projection of a 3D vector $\mathbf{X} = (X, Y, Z)$ in space (the 'scene') to 2D points (x, y), with $x = \frac{X}{Z}$ and $y = \frac{Y}{Z}$, on the image plane.
  That is: a camera takes images of the world and displays the result on an image plane.
- Projective transformation or homography
  Mapping a plane to a different plane.
  3×3 matrix $\mathbf{H}$ such that for any point in $\mathbf{X}$ it holds that: $\mathbf{X}' = \mathbf{HX}$.

$$\mathbf{H} = \mathbf{R} + \mathbf{t}\frac{\mathbf{n}^T}{d} \tag{12}$$

Homography



$$\mathbf{H} = \mathbf{R} + \mathbf{t}\frac{\mathbf{n}^T}{d}$$

Homography estimation with Direct Linear Transform

Given a set of 2D to 2D point correspondences, $\mathbf{x}_i \leftrightarrow \mathbf{x}'_i$, a perspective transformation is written as; $\mathbf{x}'_i = \mathbf{H}\mathbf{x}_i$, with $i \in \{1, 2, \ldots, p\}$. As this definition involves homogeneous vector transformation, it can be expressed in the form of a vector cross products as

$$\mathbf{x}'_i \times \mathbf{H}\mathbf{x}_i = \mathbf{0}. \tag{13}$$

Rearranging then gives and expression as $\mathbf{A}_i\mathbf{h} = \mathbf{0}$ or

$$\begin{bmatrix} 0 & 0 & 0 & -x_i & -y_i & -1 & x_i y'_i & y_i y'_i \\ x_i & y_i & 1 & 0 & 0 & 0 & -x_i x'_i & -y_i x'_i \end{bmatrix} \mathbf{h} = \mathbf{0}. \tag{14}$$

where $\mathbf{h} = [h_1\ h_2\ h_3\ h_4\ h_5\ h_6\ h_7\ h_8\ ]^T$.
A minimum of four non-collinear point matches is needed to solve this correspondence problem (more is better estimate).
Many methods to solve this: SVD, least-squares, RANSAC

Homography

How to get $\mathbf{R}$ and $\mathbf{t}$?

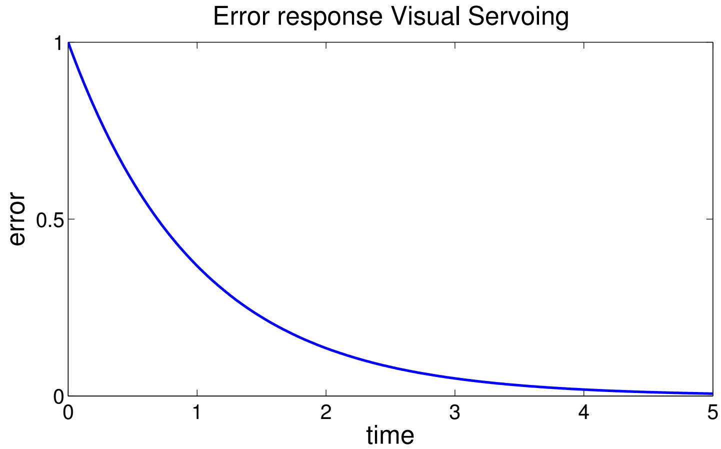
Pretty complicated calculation (SVD + orthonormal basis + ...)

- What it boils down to is a quadratic equation;
  thus eventually two solutions are feasible
- With simple heuristic pick the correct one
  - by checking normal vector
  - by checking previous output
- What does this mean?
  $\mathbf{R}$ and $\mathbf{t}$ are the errors we can use for PBVS!

Questions

Trajectory Generation

4. Trajectory Generation



- Motion of VS is not designed ($\dot{\mathrm{e}} = -\lambda \mathrm{e}$)
- Step function at start
- No constraint on time, maximum velocity, acceleration, etc.

Trajectory Generation

Basic problem:

Move the robot/quadcoptor from initial pose to end pose
(may go through some via-point)

- Path points: initial, end, via
- Trajectory: time history of position, velocity, acceleration
- Constraints: Spatial, temporal, smoothness

Trajectory Generation

Path vs. trajectory

- Path: Only geometric; the way from A to B
- Trajectory: include time, dynamics
- Kinodynamic motion planning:
  kinematics + dynamics included in motion planning

Path planning methods include:

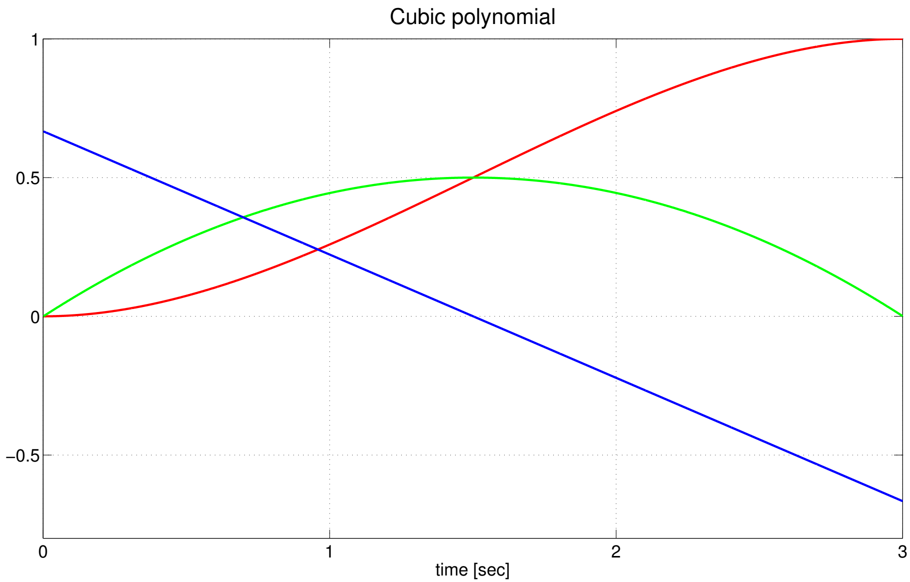
- Potential Field approaches
  Construct a force-field to 'push' and 'pull' on the robot
- Probabilistic Roadmaps (PRM)
  Generate minimalistic representation of the free space by sampling
- More focus on collision avoidance

Simple example

Move the robot/quadcoptor from 0 to 1 in 3 seconds, $\dot{q}_i = 0$ and $\dot{q}_f = 0$.

- $t_i = 0$, $t_f = 3$, $q_i = 0$, $q_f = 1$
- cubic polynomial: $q(t) = a(1) + a(2)t + a(3)t^2 + a(4)t^3$
  $\dot{q}$ = derivative of q
- solve: $\mathbf{a} = \mathbf{M}^{-1}\mathbf{b}$
  $\mathbf{M}$: Vandermonde matrix containing motion trajectory
  $\mathbf{b}$: constraint vector = $[q_i,\ q_f,\ \dot{q}_i,\ \dot{q}_f]^T$
  $\mathbf{a}$: polynomial coefficients
  t: time

Trajectory Generation



- $\dot{q}_{max} = 0.5$
- Discontinuities!

Trajectory Generation

Generalize

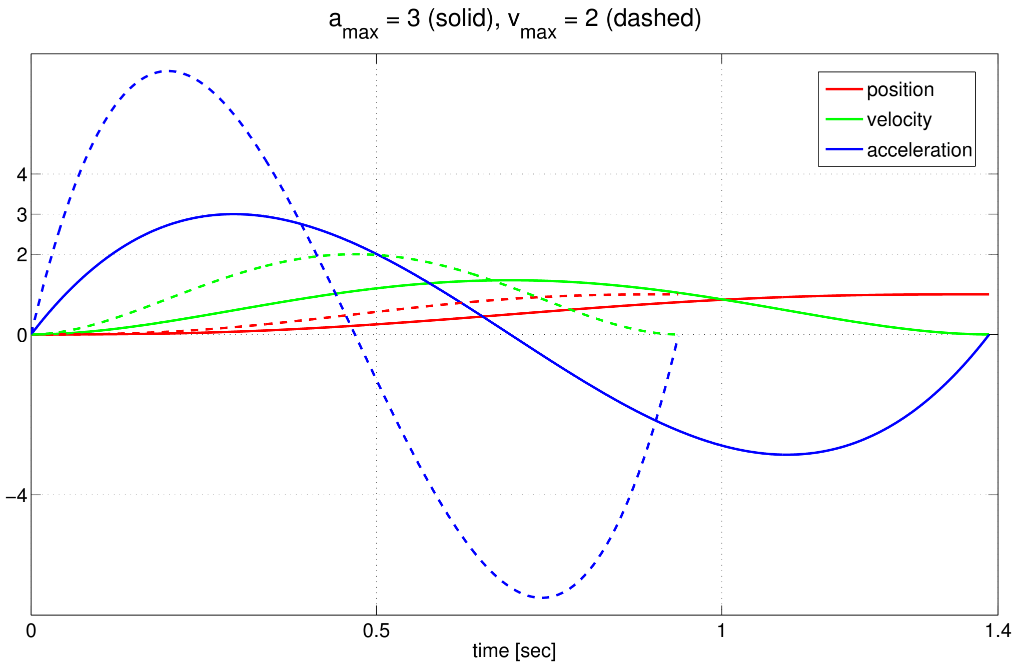
$$\mathbf{b} = \begin{bmatrix} q_0 & q_1 & \dots & q_{n-1} & q_n & v_0 & \alpha_0 & v_n & \alpha_n \end{bmatrix}^T = \mathbf{M}\mathbf{a} =$$

$$\begin{bmatrix} 1 & t_0 & & & \dots & t_0^{n+4} \\ 1 & t_1 & & & \dots & t_1^{n+4} \\ & & & & \vdots & \\ 1 & t_{n-1} & & & \dots & t_{n-1}^{n+4} \\ 1 & t_n & & & \dots & t_n^{n+4} \\ 0 & 1 & 2t_0 & & \dots & (n+4)t_0^{n+3} \\ 0 & 0 & 2 & 6t_0 & \dots & (n+4)(n+3)t_0^{n+2} \\ 0 & 1 & 2t_n & & \dots & (n+4)t_n^{n+3} \\ 0 & 0 & 2 & 6t_n & \dots & (n+4)(n+3)t_n^{n+2} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \\ a_n \\ a_{n+1} \\ a_{n+2} \\ a_{n+3} \\ a_{n+4} \end{bmatrix} \qquad (15)$$

- Multiple points, multiple constraints
- Higher order = oscillations! (Runge's phenomenon)

Scaling to incorporate max constraints

- Velocity: $t_{v,max} = \frac{15}{8}\frac{h}{v_{max}}$, $h = q_f - q_i$
- Acceleration: $t_{\alpha,max} = \sqrt{\frac{10\sqrt{3}}{3}\frac{h}{\alpha_{max}}}$

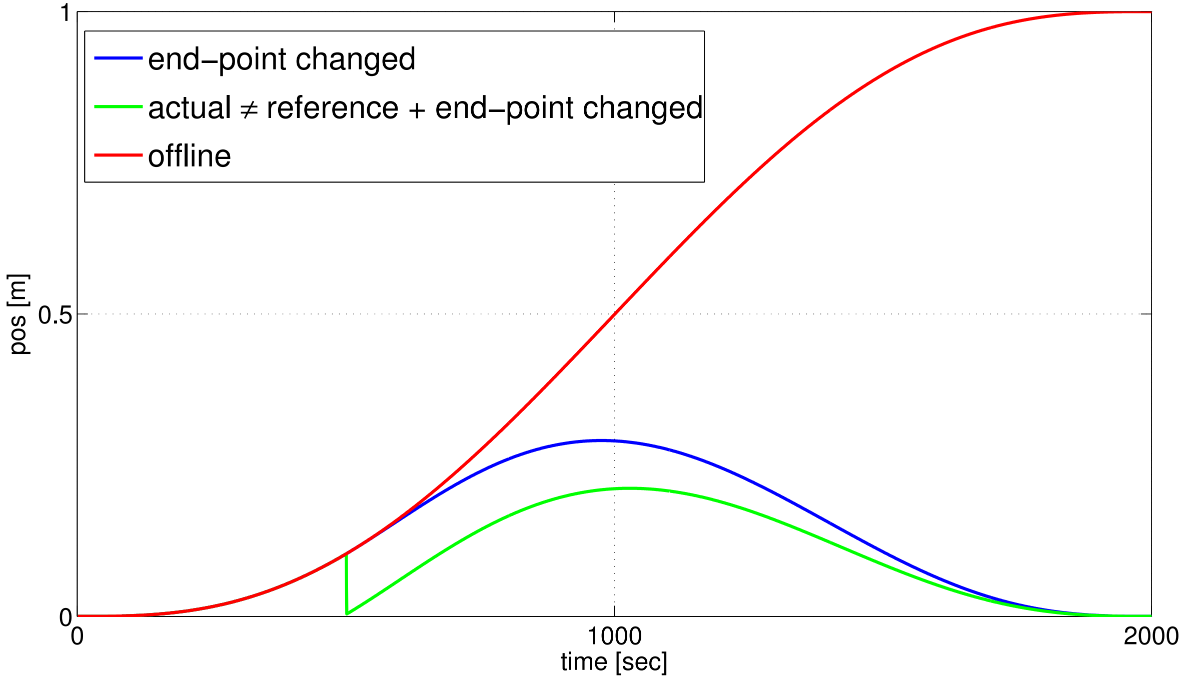


Trajectory Generation

Offline vs. Online

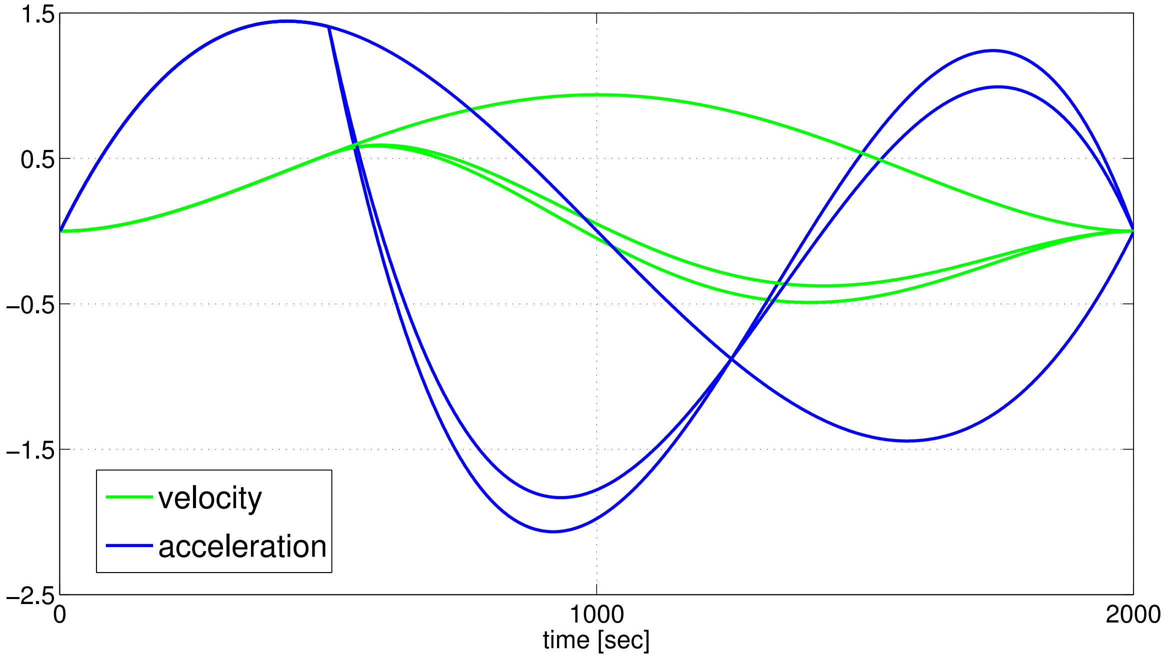
- All the previous is calculated offline
- What if..
  - Actual pose $\neq$ reference pose
  - non-static reference
- Avoid this by generating a trajectory every iteration, or every visual update

Online trajectory generation: position



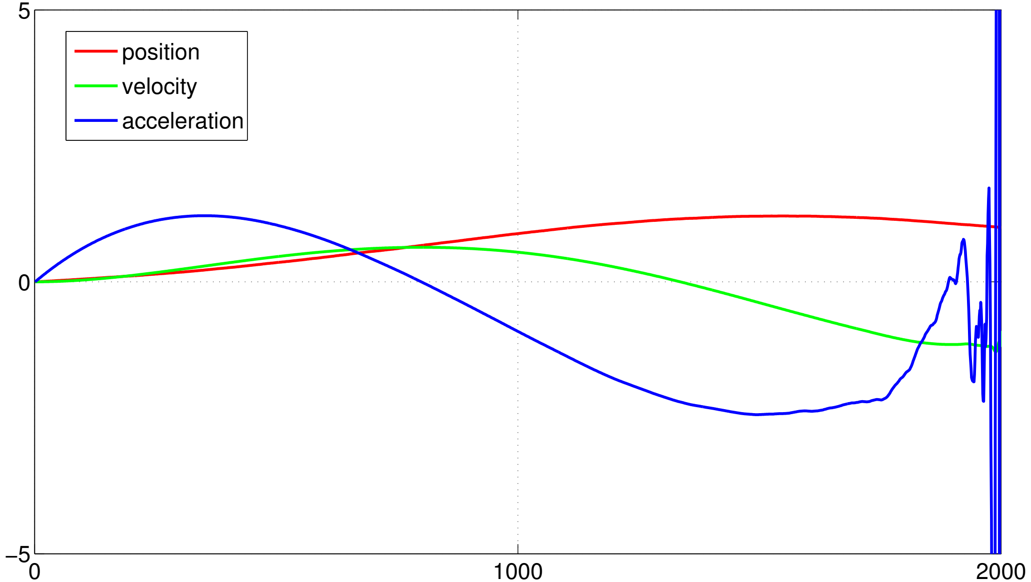
Trajectory Generation

Online trajectory generation: velocity & acceleration



Online trajectory generation: Complications

- Changing trajectory too close to the end (constraints?)
- Not enough time to apply changes: solve by adding more time



Big picture

5. Putting it all together

- Robot: Quadcoptor
  - Local control: to stabilize
  - Global control: for vision-based motion
- Vision based control
  - IBVS with simple features
  - Pose-error from SURF + homography + decomposition
- Trajectory
  - Error minimization: 'traditional' VS
  - Online TG: design point-to-point/multipoint with constraints

Questions

References

[1] F. Chaumette, and S. Hutchinson, "Visual Servo Control Part I: Basic Approaches", in *IEEE Robotics and Automation Magazine*, vol. 13(4), pp. 82-90, 2006.

[2] R. Hartley, and A. Zisserman, "Multiple View Geometry in Computer Vision", Cambridge University Press, 2003.

[3] Y. Ma, S. Soatto, J. Kosecka, and S. Sastry, "An Invitation to 3D Vision. From Images to Geometric Models", Springer-Verlag, 2004.

[4] Herbert Bay, Andreas Ess, Tinne Tuytelaars, Luc Van Gool, "SURF: Speeded Up Robust Features", Computer Vision and Image Understanding (CVIU), Vol. 110, No. 3, pp. 346–359, 2008

[5] Trajectory Planning for Automatic Machines and Robots, Luigi Biagiotti, Claudio Melchiorri

See course website for more references.