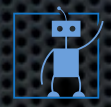


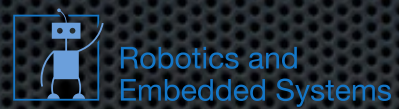
Applied Computer Vision for Robotics

7.4.2014

Philipp Heise (heise@in.tum.de)

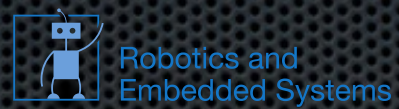
Brian Jensen (jensen@in.tum.de)

GitHub



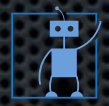
- Find together in groups of 3 people
- Everyone needs to create a GitHub account http://github.com
- Write a mail to jensen@in.tum.de containing:
  - team name (be creative here)
  - real name, GitHub account name and e-mail address for each team member

Syllabus



- New sheet every 2 weeks
- **Mandatory** meeting every week
- Short presentation of the sheet results by the teams
- Everyone will be registered to a RVC mailing list - ask questions here
- You can also ask us questions or write mails

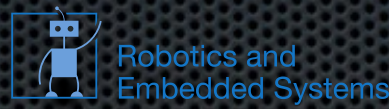
ROS



Robotics and
Embedded Systems

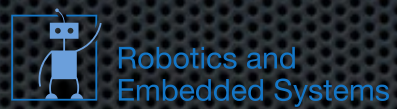
- We are going to use ROS - Robot Operating System (www.ros.org)
- We'll introduce it on the fly and give hints which packages you might need
- Check out the tutorials and wiki on ros.org for more details

Cameras



- If you don’t have an own camera working in ROS, you can borrow one from us
- PSEye Camera - 640x480 @60Hz
- **Deposit: 10 €**

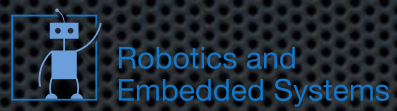
Sheet 1



- Task: detect features in images
- Features are hopefully distinctive regions in the image e.g. corners or lines that can be found with high repeatability
- Very useful to find correspondences between images

[1] R. Szeliski, Computer Vision: Algorithms and Applications, 1st ed. New York, NY, USA: Springer-Verlag New York, Inc., 2010.

Harris corners

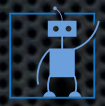


- We want to know how much the image changes around our current pixel
- Taylor expansion for image

$$I(\mathbf{x} + \Delta\mathbf{u}) \approx I(\mathbf{x}) + \nabla I(\mathbf{x}) \cdot \Delta\mathbf{u}$$

[1] C. Harris and M. Stephens, “A combined corner and edge detector.,” vol. 15, p. 50, 1988.

Harris corners



$$E(\Delta\mathbf{u}) = \sum_i w(\mathbf{x}_i)(I(\mathbf{x}_i + \Delta\mathbf{u}) - I(\mathbf{x}_i))^2$$

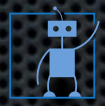
$$\approx \sum_i w(\mathbf{x}_i)(I(\mathbf{x}_i) + \nabla I(\mathbf{x}_i) \cdot \Delta\mathbf{u} - I(\mathbf{x}_i))^2$$

$$= \sum_i w(\mathbf{x}_i)(\nabla I(\mathbf{x}_i) \cdot \Delta\mathbf{u})^2$$

$$= \Delta\mathbf{u}^\top \mathbf{A} \Delta\mathbf{u}$$

[1] R. Szeliski, Computer Vision: Algorithms and Applications, 1st ed. New York, NY, USA: Springer-Verlag New York, Inc., 2010.

Harris corners



- With the matrix:

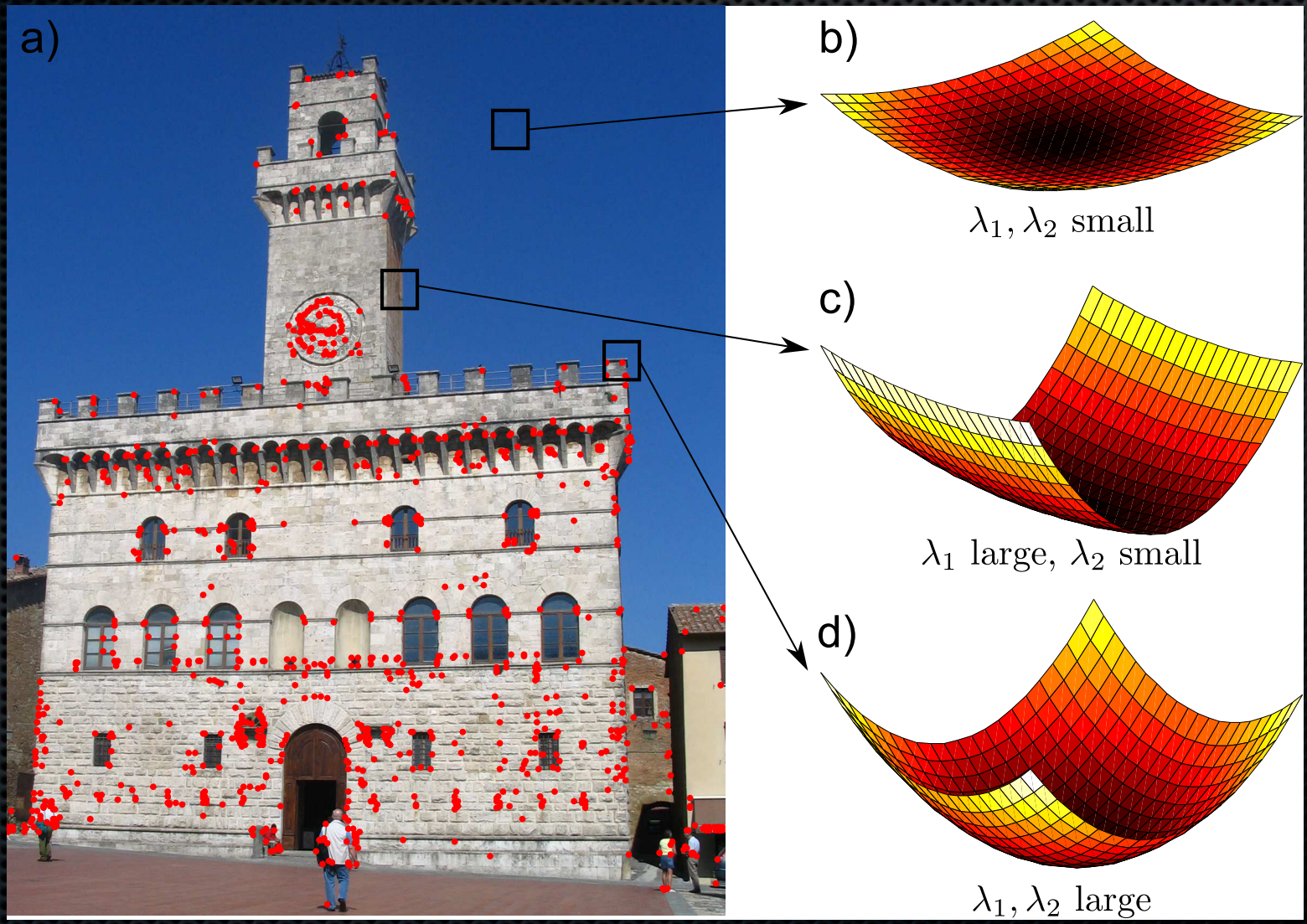
$$A = \sum_i w(\mathbf{x}_i) \begin{pmatrix} \nabla_x I(\mathbf{x}_i)^2 & \nabla_x I(\mathbf{x}_i) \nabla_y I(\mathbf{x}_i) \\ \nabla_x I(\mathbf{x}_i) \nabla_y I(\mathbf{x}_i) & \nabla_y I(\mathbf{x}_i)^2 \end{pmatrix}$$

- We are only interested in the two eigenvalues of the matrix in order to find a certain type of patch

[1] R. Szeliski, Computer Vision: Algorithms and Applications, 1st ed. New York, NY, USA: Springer-Verlag New York, Inc., 2010.

Harris

Robotics and Embedded Systems



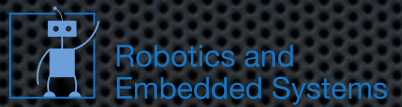
Harris corners

- The autocorrelation matrix is enough to know how much the intensity changes around our current pixel
- We can use the eigenvalues of the matrix to classify if the area is flat, edgy or a corner
- Harris and Stephens proposed ($\alpha \approx 0.06$)

$$\det(\mathbf{A}) - \alpha \operatorname{trace}(\mathbf{A})^2 = \lambda_0 \lambda_1 - \alpha(\lambda_0 + \lambda_1)^2$$

to find regions where both eigenvalues are large

Harris corners



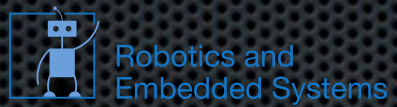
- Calculate structure tensor

$$A = \sum_i w(\mathbf{x}_i) \begin{pmatrix} \nabla_x I(\mathbf{x}_i)^2 & \nabla_x I(\mathbf{x}_i) \nabla_y I(\mathbf{x}_i) \\ \nabla_x I(\mathbf{x}_i) \nabla_y I(\mathbf{x}_i) & \nabla_y I(\mathbf{x}_i)^2 \end{pmatrix}$$

- Be careful about the image data types
- Use the eigenvalues λ_1, λ_2 to calculate the "corneress"

[1] R. Szeliski, Computer Vision: Algorithms and Applications, 1st ed. New York, NY, USA: Springer-Verlag New York, Inc., 2010.

Harris corners



- Use the Harris-Stephens formula to avoid the exact computation of the eigenvalues

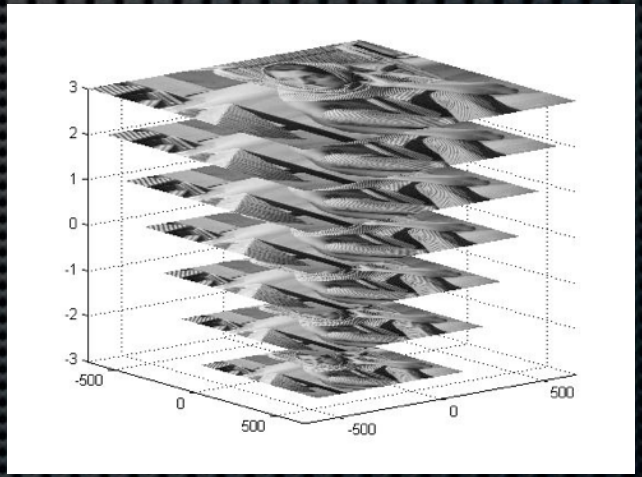
$$M_c = \lambda_1 \lambda_2 - \kappa(\lambda_1 + \lambda_2)^2$$
$$= \det(M) - \kappa \, \mathrm{trace}^2(M)$$

- Kappa is a parameter and can be tuned to get reasonable results
- Threshold to get corners and perform non-maximum suppression

[1] R. Szeliski, Computer Vision: Algorithms and Applications, 1st ed. New York, NY, USA: Springer-Verlag New York, Inc., 2010.

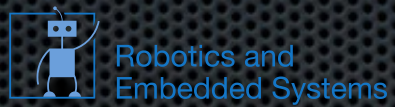
Scale Space

- Perform Harris corner detection on different scales to get scale invariance
  - Create an image pyramid with a scale factor (<1) and a fixed number of octaves
  - Detect the corners in each octave



[1] R. Szeliski, Computer Vision: Algorithms and Applications, 1st ed. New York, NY, USA: Springer-Verlag New York, Inc., 2010.

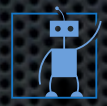
Orientation



- Find the orientation of each corner to get rotation invariance
- Compute the angle using

$$\theta = \mathrm{atan2}\left(\sum_i w(\mathbf{x}_i)\begin{pmatrix}\nabla_x I(\mathbf{x}_i)\\ \nabla_y I(\mathbf{x}_i)\end{pmatrix}\right)$$

NMS and ANMS

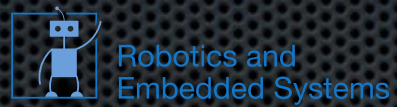


- Very often neighbouring pixels are all classified as corners - suppress the non maximum responses (NMS)
- Very often the detected features are also not well distributed over the image
- Instead of using a fixed threshold ANMS uses only non

[1] M. Brown, R. Szeliski, and S. Winder, “Multi-image matching using multi-scale oriented patches,” vol. 1, pp. 510–517, 2005.

[2] R. Szeliski, Computer Vision: Algorithms and Applications, 1st ed. New York, NY, USA: Springer-Verlag New York, Inc., 2010.

Visualization



- For the purpose of debugging it is highly recommended to draw the detected features (e.g. as circles)
- Represent the scale by the size of the radius
- Draw the orientation as an line inside the circle