Lab Course: Robot Vision WS 2012/2013

Philipp Heise, Brian Jensen, Sebastian Klose

Assignment 1 - Due: 05.11.2012

Exercise 1 Getting Started with ROS

1. *(ROS Tutorials)* The best way for getting started with ROS, is using the Tutorials provided on http://www.ros.org/wiki/ROS/Tutorials
   Walk through all the Tutorials on this page, to get an overview about the ROS system. As C++ will be the language of choice throughout this course, pleace stick to those tutorials where needed.
2. *(Image Publishing & Subscription)* By now you should have a basic understanding of how to create your own package within ROS. Your first task will be to write some simple image publishing and subscription nodes. First create a new package named imagepub_`<group>`. Make sure you add the following ROS packages as dependencies when you create your new ROS package:

   `roscpp`, `image_transport`, `cv_bridge`, `dynamic_reconfigure`.
   Implement the following nodes:

   - A node named `cb_publisher`. The node shall advertise an image topic named `cb_img`, containing a generated image of a checkerboard. Your node should have the following ROS parameters:
     - `width` - the width of the advertised image
     - `height` - the height of the advertised image
     - `square_size` - the size in pixels of one square of the checkerboard
     - `frequency` - the frequency with which to publish the checkerboard image

     You should also give useful default values for these parameters. The following wiki pages might be useful for this taks:
     - http://www.ros.org/wiki/roscpp/Overview/Parameter%20Server#Getting_Parameters
     - http://www.ros.org/wiki/image_transport/Tutorials/PublishingImages
     - http://ros.org/doc/api/sensor_msgs/html/msg/Image.html

     *Note: For exercise you are **NOT** allowed to use any opencv functions, including the OpenCV bridge. Instead you should create and manipulate the image message directly.*
   - A node named `file_publisher`, that loads an image from disk and advertises it under the topic `image`. Here you should use OpenCV for loading the image and the `cv_bridge` for converting an OpenCV image into a ROS image message. Your node should have the following parameters:
     - `file` - the path to the image to load
     - `frequency` - frequency used for publishing the image (use a default parameter here)

Related wiki pages:

- http://www.ros.org/wiki/cv_bridge
- http://docs.opencv.org/ - imread function

To visualize your results you can use the `image_view` node in the `image_view` package. See http://www.ros.org/wiki/image_view for information on its usage.

4. *(Image Pipeline)* In this exercise you will implement your first image processing pipeline in ROS. You will implement a node that subscribes to an image topic, performs some simple image manipulation, then publishes the resulting image on a separate image topic. Specifically the new node will perform image brightness and contrast adjustment.

   - Create a new node in the imagepub_`<group>` package named `image_changer`. Create a new class named `ImageChanger`. The new node should have the following ROS parameters:
     - `brightness` - an integer value greater than or equal to zero.
     - `contrast` - a floating point value greater than zero.
   - The first task is to make the new node subscribe to an image topic. Use the `subscribe` method of the `ImageTransport` class to receive a callback in a member function of the class.
   - Next you should perform brightness and contrast adjustment on incoming image. This is accomplished using a simple linear operation on the raw image data according to the formula:

     $$I(x,y)' = c * I(x,y) + b$$

     where b the brigheness change is, c the constrast scaling factor, and $I(x,y)$ the image itensity value at position x, y. *Note: For multichannel images, i.e RGB, this operation can be performed on each channel individually.*
   - Publish the resulting image message on a new image topic.

   Visualize the resulting image with `image_view` to verify your new node operates correctly. *Note: For exercise you are **NOT** allowed to use any OpenCV functions, including the OpenCV bridge. Instead you should create and manipulate the image message directly.*

5. *(Dynamic Reconfigure)* Now its time to get to know a very useful piece of ROS infrastructe: dynamic reconfigure. Here you are going to make the `image_changer` node respond to parameter changes at runtime using the dynamic reconfigure infrastructure.

   - Create a new directory named `cfg` in the imagepub_`group` package. Create a new dynamic configuration file in that directory named `ImageChangerConfig.cfg` and make this file executable. The new configuration file should contain the appropriate entries for the `image_changer` node.
   - Make the appropriate changes to the `CMakeLists.txt` file for dynamic reconfigure.
   - Modify the `ImageChanger` class to receive a callback in a member function from the dynamic reconfigure `Server` class.

   Test your implementation using the `reconfigure_gui` in the `dynamic_reconfigure` package.

Exercise 2 Harris Corner Detection

In this exercise, we will implement a node that computes the harris corners for an input image stream. The node will subscribe to an image message topic and publish a resulting image topic.

1. *(Image Subscription)* Create a package named harris_¡group¿. Consider which dependencies are appropriate for the new harris corner detector package.
   - In this package create a node named `harris_corners`.
   - Create a class that subscribes to an input image topic and receives the callbacks in a member function. Use the OpenCV `cv_bridge` functionality for creating an OpenCV image (`cv::Mat`) from the incoming image message.
   - Convert the OpenCV image to grayscale using the appropriate OpenCV color conversion function.

2. *(Parameters)* The `harris_corners` node should to support the following parameters:
   - `template_size` - an odd integer value greater than or equal to three that contains the size of the template for calculating the harris response at a given point.
   - `threshold` - a floating point value greather than zero that sets the lower boundary for a harris response value to be considered a corner. This value is in percent and taken relative to the maximum harris response in the input image, i.e. a value of 0.1 means that all response values greater than 10% of the maximum response are considered corners.

3. *(Harris corner detection)* Recall that the Harris corner detector looks for points in the image where the immediate area enclosing the point has high self dissimilarity in all directions. This can be approximately estimated using image partial derivatives and the *Sum of Squared Differences* method:

$$S(x, y; \Delta x, \Delta y) = [\Delta x \Delta y]\, Q(x, y) \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}$$

where $S(x, y; \Delta x, \Delta y)$ is the approximated sum of sqaured differences between the image patch at (x, y) and at $(x + \Delta x, y + \Delta y)$ and where

$$Q(x, y) = \sum_{(u,v) \in W(x,y)} w(u, v) \begin{bmatrix} I_x(u, v)^2 & I_x(u, v) I_y(u, v) \\ I_x(u, v) I_y(x, y) & I_y(u, v)^2 \end{bmatrix} = \begin{bmatrix} A & B \\ B & C \end{bmatrix}$$

is the Harris matrix at (x, y). This matrix contains a sum of the squared image parital derivates over the template window $W(x, y)$ centered at the image location, where each partial derivative term is weighted arrording to the template $w(u, v)$ (which can either be a guassian or a contant factor). Harris corners are characterized by locations which have two large eigenvalues $\lambda_1 \lambda_2$ in the Harris matrix. Instead of performing eigen value decomposition we will use the alternative formulation for checking this trait proposed by Harris:

$$\lambda_1 \lambda_2 - 0.04(\lambda_1 - \lambda_2)^2 = AC - B^2 - 0.04(A + C) \tag{1}$$

which is known as the Harris response for a point.

   - Create a member function for computing the Harris corner response that takes a **graysacale** OpenCV image as input and returns an image containing the Harris response at every valid point (the template does not exceed the image boundaries at the point).
   - Calculate the matrices A, B, and C for each valid point:

$$A(x, y) = \sum_W I_x(x, y)^2, \quad B(x, y) = \sum_W I_x(x, y) I_y(x, y), \quad C(x, y) = \sum_W I_y(x, y)^2$$

   - Calculate the Harris reponse at each valid point using equation 1. Determine the maximum harris response value and use the `threshold` parameter to determine which image points are Harris corners.

- Create an ouput image where each of the detected Harris corners is drawn on the input image using an appropriate OpenCV drawing function. Publish the output image on a separate image topic.

*You may **NOT** use the OpenCV corner detection! However, you may use the Sobel, Gaussian or Convolution filters as well as any of the drawing functions.*

4. *(Dynamic reconfigure)* Rework your Harris implementation to use dynamically reconfigurable parameters. Use the `reconfigure_gui` to visualize how changes of the parameters affects your corner detector implementation.

Exercise 3 Nodelets

In this exercise you will get to know Nodelets. Nodelets are a piece of ROS infrastructure to enable minimal overhead communication between nodes. Nodelets are basically equivalent to nodes, implemented as threads rather than full-blown processes. A nodelet manager serves as the thread pool process. Nodelets can be loaded dynamically into running managers, as they are implemented as plugins. Read the relevant wiki pages about Nodelets for more detailed informaition.

1. *(Harris Nodelet)*
   - write an additional nodelet version of your existing harris implementation
   - create a launch file, that brings up a manager and your nodelet

Exercise 4 Visualization in RViz

1. *(Images and Poses)* Download the bagfile from (`wget` http://www6.in.tum.de/~kloses/rvc/kitty_01.bag). Playback the content of the file using the `rosbag` tool. Use `rviz` to visualize the images in the bagfile (image Display). The bag file also contains the pose of the moving camera, which is has been published as a `tf` frame. Use a `tf` Display in RViz, to visualize this pose, such that the camera is the moving frame.
2. *(Marker Visualization)* You can also send custom entities to rViz, to enhance the visualization for your needs. You can find the details about Marker-Visualization on http://www.ros.org/wiki/rviz/Tutorials. For the moment, we are only interested in normal markers (first two tutorials).

   - Create a rospackage called `rviz_markers_<group>`. Implement a node called `rviz_clock` visualizing the current time as analog clock in rviz.

3. *(Moving Model)* Using the Mesh-Type marker, you can specify a model file, which shall be used for visualization. Write a node, to visualize our chair logo within rviz. Make the logo follow the `tf` frame of the moving camera in the bagfile. You can download the logo at: http://www6.in.tum.de/~kloses/rvc/logo.dae

Exercise 5 Camera Calibration

In order to get metric information out of images, we need to calibrate our cameras for the intrinsic parameters. ROS provides a tool for that. First bring up the camera node using the `uvc_camera` package. You probably need to specify a namespace for the camera on startup. (*ROS_NAMESPACE=cam rosrun ...*)

1. *(Monocular camera calibration)* Follow the tutorial on http://www.ros.org/wiki/camera_calibration/Tutorials/MonocularCalibration and calibrate one of the lab cameras using the provided calibration pattern. *Commit* your results before closing the camera. Now verify if your results have been correctly stored within your camera, by first inspecting the corresponding camera_info message and then starting up the image_proc node

of ROS (see http://www.ros.org/wiki/image_proc for details). *Note: Make sure to make a backup copy of the save calibration file somewhere in your repository*