Processing Camera Data

ROS + PR2 Training Workshop
Outline

• Cameras on the PR2
• The monocular image pipeline
• The stereo image pipeline
• Logging sensor data
• Writing a vision node
Outline

- Cameras on the PR2
  - The camera suite
  - Viewing images
  - Adjusting camera parameters
  - Using the texture projector
  - Saving bandwidth
- The monocular image pipeline
- The stereo image pipeline
- Logging sensor data
- Writing a vision node
Cameras on the PR2

- Wide angle arm camera
- Wide angle color stereo
- LED texture projector
- Narrow angle mono stereo
- 5MP camera
Viewing Images

• rviz node
  • Displays panel → Add → Camera

• image_view node
  • $ rosrun image_view image_view
    image:=<camera>/<image>
Configuring Your Cameras

$ rosrun dynamic_reconfigure reconfigure_gui

For cameras:

- Adjust camera parameters (exposure, gain, ...)
- Turn texture projector on/off
- Adjust stereo processing parameters

www.ros.org/wiki/dynamic_reconfigure
Adjust Camera Parameters

Cameras:
- `/l_forearm_cam`
- `/r_forearm_cam`
- `/wide_stereo_both`
- `/narrow_stereo_both`
- `/prosilica_driver`

For stereo cameras, “both” propagates settings to left & right
Using the Texture Projector

- projector_mode – whether projector is turned on

- *_trig_mode – whether the camera synchs with the projector on all, no, or some frames

- Camera namespaces change when using the texture projector:
  - /narrow_stereo
  - /narrow_stereo_textured
Saving Bandwidth

• Each image topic has alternate compressed versions

• Transports available out of the box
  • “raw” – default, uncompressed
  • “compressed” – JPEG or PNG
  • “theora” – Theora video codec

• With image_view:
  $ rosrun image_view image_view
  image:=<camera>/<image> compressed

www.ros.org/wiki/image_transport
Outline

- Cameras on the PR2
  - The monocular image pipeline
    - Camera calibration
    - Basic processing
      - De-Bayering
      - Rectification
  - The stereo image pipeline
- Logging sensor data
- Writing a vision node
The Monocular Image Pipeline

Camera Driver

<camera>/image_raw
<camera>/camera_info

image_proc

<camera>/image_rect_color

image_view
rviz
Your vision node

Images

Settings

Mono Camera Calibration

- Parameters → camera driver
- CameralInfo message published with each Image

www.ros.org/wiki/camera_calibration
Mono Camera Calibration

- Get a large checkerboard

- $ rosr$un camera_calibration
cameracalibrator.py --size 8x6 --square 0.108 image:=/camera/image_raw
camera:=/camera

www.ros.org/wiki/camera_calibration/Tutorials/MonocularCalibration
Mono Processing

image_proc publishes image topics that are
- De-bayered (grayscale or color)
- Rectified

On topics:
<camera>/image_mono
<camera>/image_color
<camera>/image_rect
<camera>/image_rect_color

www.ros.org/wiki/image_proc
Outline

✓ Cameras on the PR2
✓ The monocular image pipeline

• The stereo image pipeline
  • Stereo calibration
  • Stereo processing (3D)
  • Viewing disparity images and point clouds
  • Adjusting stereo parameters

• Logging sensor data
• Writing a vision node
The Stereo Image Pipeline

Images

Settings

Camera Driver

<camera>/left/image_raw

<camera>/left/camera_info

Stereo Camera Calibration

- Distortion
- Stereo rectification
- 3D geometry

- Parameters → camera drivers
- CameralInfo message published with each Image

www.ros.org/wiki/camera_calibration
Stereo Processing

stereo_image_proc publishes

• Disparity images
• Point clouds

On topics:
<stereo_camera>/disparity
<stereo_camera>/points

www.ros.org/wiki/stereo_image_proc
Viewing Disparity Images

$ rosrun image_view stereo_view
  stereo:=narrow_stereo  image:=image_rect
Viewing Point Clouds

Displays panel → Add → Point Cloud

Enter the topic in the red box
Adjust Stereo Parameters

In reconfigure_gui, look for <stereo>/stereo_image_proc

www.ros.org/wiki/stereo_image_proc/Tutorials/ChoosingGoodStereoParameters
Exercise #1

Calibrate your narrow stereo cameras.

www.ros.org/wiki/camera_calibration/Tutorials/SteroCalibration

But do not “Commit” to the camera, or you will have to run full-body calibration again!
Outline

- Cameras on the PR2
- The monocular image pipeline
- The stereo image pipeline

- Logging sensor data
  - Recording and playback
  - Visual inspection with rxbag
  - Bags and ROS time

- Writing a vision node
Logging Sensor Data

Recording data:
$ rosbag record r_forearm_cam/image_raw r_forearm_cam/camera_info tf

Play back data:
$ rosbag play XXX.bag

What's in a bag file:
$ rosbag info mystery_data.bag

www.ros.org/wiki/rosbag/Tutorials/Recording and playing back data
Visual Inspection with rxbag

$ rxbag XXX.bag

View thumbnails:
Right-click → Thumbnails... → select topic(s)

Image viewer for a topic:
Right-click → View (by datatype)... → sensor_msgs/Image → <topic> → Image

www.ros.org/wiki/rxbag
Bags and ROS Time

- Do not play back a bag against a live robot!

- Recorded message timestamps will be far in the past relative to “wall-clock” time

- When using time-aware nodes with bagged data:
  - `rosbag play --clock XXX.bag`
  - Set parameter `/use_sim_time = True` before starting nodes

- Do not set `/use_sim_time` on a live robot!

www.ros.org/wiki/Clock
Outline

- Cameras on the PR2
- The monocular image pipeline
- The stereo image pipeline
- Logging sensor data

- Writing a vision node
  - Subscribing to camera topics
  - Interfacing with OpenCV
  - Publishing images
Processing Images in ROS

Your Node

OpenCV image (cv_bridge)
OpenCV camera info (image_geometry)

Solve vision

Publish results

www.ros.org/wiki/cv_bridge
www.ros.org/wiki/image_geometry
Subscribing to Camera Topics

Use `image_transport` instead of `ros::Subscriber`
- Compression (JPEG/PNG, Theora)
- Add others as plugins

Subscribe to image AND camera_info topics
- Synchronized time stamps
- `CameraSubscriber` handles the synchro

www.ros.org/wiki/image_transport
Subscribing to Camera Topics

```cpp
#include <ros/ros.h>
#include <image_transport/image_transport.h>

class MyVisionNode {
  ros::NodeHandle nh_;  
  image_transport::ImageTransport it_;  
  image_transport::CameraSubscriber sub_;  

public:
  MyVisionNode() : it_(nh_) {
    sub_ = it_.subscribeCamera("image_topic", 1, 
  &MyVisionNode::imageCb, this);
  }

  void imageCb(const sensor_msgs::ImageConstPtr& image_msg,
  const sensor_msgs::CameraInfoConstPtr& info_msg) {
    // ...
  }
};
```
Open Computer Vision Library

OpenCV Overview: > 500 functions

General Image Processing Functions

Image Pyramids

Segmentation

Geometric descriptors

Camera calibration, Stereo, 3D

Transforms

Features

Utilities and Data Structures

Machine Learning:
- Detection,
- Recognition

Tracking

Matrix Math

Fitting

opencv.willowgarage.com
Using ROS messages with OpenCV

cv_bridge
- ROS sensor_msgs/Image → OpenCV IplImage

image_geometry
- ROS sensor_msgs/CameraInfo → OpenCV calibration matrices

Many useful functions in the camera model classes

www.ros.org/wiki/cv_bridge/Tutorials
www.ros.org/wiki/image_geometry
ROS Image -> OpenCV

```cpp
#include <cv_bridge/CvBridge.h>

class MyVisionNode {

    sensor_msgs::CvBridge bridge_;

public:
    void imageCb(const sensor_msgs::ImageConstPtr& image_msg,
                 const sensor_msgs::CameraInfoConstPtr& info_msg) {
        IplImage *cv_image = NULL;
        try {
            cv_image = bridge_.imgMsgToCv(image_msg, "bgr8");
        } catch (sensor_msgs::CvBridgeException& error) {
            ROS_ERROR("Couldn't convert image with encoding %s",
                      image_msg->encoding.c_str());
            return;
        }
    }
};
```
ROS CameraInfo -> OpenCV

```cpp
#include <image_geometry/pinhole_camera_model.h>

class MyVisionNode {
public:
    image_geometry::PinholeCameraModel cam_model_; 

    void imageCb(const sensor_msgs::ImageConstPtr& image_msg, 
                  const sensor_msgs::CameraInfoConstPtr& info_msg) 
    {
        cam_model_.fromCameraInfo(info_msg);
    }
};
```
Publishing Image Topics

```cpp
1 class MyVisionNode
2 {
3   ros::NodeHandle nh_;  
4   image_transport::ImageTransport it_;  
5   image_transport::Publisher pub_;  
6
7 public:
8   MyVisionNode()
9     : it_(nh_)  
10  {
11     pub_ = it_.advertise("image_out", 1);
12  }
13
14   void imageCb(const sensor_msgs::ImageConstPtr& image_msg,  
15                  const sensor_msgs::CameraInfoConstPtr& info_msg)
16  {
17     // ...
18     pub_.publish(bridge_.cvToImgMsg(image, "bgr8"));  
19  }
20 
```

```
Outline

✓ Cameras on the PR2
✓ The monocular image pipeline
✓ The stereo image pipeline
✓ Logging sensor data
✓ Writing a vision node
Exercise #2

Draw the location of the robot gripper (according to tf) on an image stream.

http://www.ros.org/wiki/image_geometry/Tutorials/ProjectTfFrameToImage

The tutorial uses a bag as the data source, so remember:

- Do not play back a bag against a live robot!
- Do not set /use_sim_time on a live robot!
Questions?

http://www.ros.org/

http://opencv.willowgarage.com/

ros-users@code.ros.org