Real-time Feature Tracker Library Reference Manual

Vilas K. Chitrakaran

Thu Sep 14 20:30:15 2006

Contents

Chapter 1

Real-time Feature Tracker Library Hierarchical Index

1.1 Real-time Feature Tracker Library Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Real-time Feature Tracker Library Class Index

2.1 Real-time Feature Tracker Library Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

Real-time Feature Tracker Library File Index

3.1 Real-time Feature Tracker Library File List

Here is a list of all files with brief descriptions:

Chapter 4

Real-time Feature Tracker Library Class Documentation

4.1 _feature Struct Reference

A feature point.

`#include <TrackerUtils.hpp>`

Public Attributes

- float x

  x coordinate of feature in the image.

- float y

  y coordinate of feature in the image.

- int val

  FeatureTracker classes set this to 0 if feature is tracked, else -1.

4.1.1 Detailed Description

A feature point.

4.1.2 Member Data Documentation

4.1.2.1 float _feature::x

x coordinate of feature in the image.

4.1.2.2 float _feature::y

y coordinate of feature in the image.

4.1.2.3 int _feature::val

FeatureTracker classes set this to 0 if feature is tracked, else -1.

The documentation for this struct was generated from the following file:

- TrackerUtils.hpp

4.2 _feature_list Struct Reference

List of features in a single image.

`#include <TrackerUtils.hpp>`

Public Member Functions

- _feature_list ()

Public Attributes

- int frame_number

  Image frame number correspoding to the list.

- int num_features

  Maximum number of features the list can hold.

- feature_t * features

  Array of features.

4.2.1 Detailed Description

List of features in a single image.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 _feature_list::_feature_list () `[inline]`

4.2.3 Member Data Documentation

4.2.3.1 int _feature_list::frame_number

Image frame number correspoding to the list.

4.2.3.2 int _feature_list::num_features

Maximum number of features the list can hold.

4.2.3.3 feature_t* _feature_list::features

Array of features.

The documentation for this struct was generated from the following file:

- TrackerUtils.hpp

4.3 _FeatureServerContext Struct Reference

Parameters for UDP feature server. Use with class FeatureServer.

`#include <FeatureClientServer.hpp>`

Public Attributes

- int port
- int thread_priority
- int num_features

4.3.1 Detailed Description

Parameters for UDP feature server. Use with class FeatureServer.

4.3.2 Member Data Documentation

4.3.2.1 int _FeatureServerContext::port

Server port number.

4.3.2.2 int _FeatureServerContext::thread_priority

Priority of the server thread.

4.3.2.3 int _FeatureServerContext::num_features

Max. number of features to serve.

The documentation for this struct was generated from the following file:

- FeatureClientServer.hpp

4.4 _FeatureTrackerContext Struct Reference

Parameters for feature tracking.

`#include <TrackerUtils.hpp>`

Public Attributes

- int num_features
- int num_frames
- int auto_select_features
- int display_tracked_features

4.4.1 Detailed Description

Parameters for feature tracking.

4.4.2 Member Data Documentation

4.4.2.1 int _FeatureTrackerContext::num_features

Max. number of features to track.

4.4.2.2 int _FeatureTrackerContext::num_frames

Number of frames to track features over.

4.4.2.3 int _FeatureTrackerContext::auto_select_features

1: if you want the tracker to select features automoatically, else 0.

4.4.2.4 int _FeatureTrackerContext::display_tracked_features

1: if you want to display tracked features, else 0.

The documentation for this struct was generated from the following file:

- TrackerUtils.hpp

4.5 _OCVTrackingContext Struct Reference

Parameters specific to OpenCV tracker (for use with FeatureTrackerOCV class).

`#include <FeatureTrackerOCV.hpp>`

Public Attributes

- int min_dist
- double quality
- int block_size
- int max_iter
- double epsilon
- int window_size
- float max_error

4.5.1 Detailed Description

Parameters specific to OpenCV tracker (for use with FeatureTrackerOCV class).

Good default values for parameters are given in parenthesis.

4.5.2 Member Data Documentation

4.5.2.1 int _OCVTrackingContext::min_dist

Minimum distance between detected corners (10).

4.5.2.2 double _OCVTrackingContext::quality

Multiplier for the maxmin eigenvalue; specifies minimal accepted quality of image corners. (0.01).

4.5.2.3 int _OCVTrackingContext::block_size

Size of the averaging block, passed to underlying cvCornerMinEigenVal() (3).

4.5.2.4 int _OCVTrackingContext::max_iter

Maximum number of iterations (20).

4.5.2.5 double _OCVTrackingContext::epsilon

Desired tracking accuracy (0.03).

4.5.2.6 int _OCVTrackingContext::window_size

Size of search window (10).

4.5.2.7 float _OCVTrackingContext::max_error

Difference between patches around the original and moved points. Should be a large value for scenes with substantial motion (200 for static camera).

The documentation for this struct was generated from the following file:

- FeatureTrackerOCV.hpp

4.6 _PXCContext Struct Reference

Parameters for the PXC200AF framegrabber. Use with class PXCCaptureLoop.

`#include <PXCCaptureLoop.hpp>`

Public Attributes

- int board_number

  -1 to request any available

- int video_channel

  Video channel number, see pxc.h.

- int pixel_format

  see frame.h for image data types. Use PBITS_Y8 or PBITS_RGB24

- int video_format

  see pxc.h for video detect types. Use NTSC_FORMAT

- int trigger_channel

  -1 for no external triggering; else triggers on rising edge.

- int thread_priority

  Priority of image capturing thread.

4.6.1 Detailed Description

Parameters for the PXC200AF framegrabber. Use with class PXCCaptureLoop.

4.6.2 Member Data Documentation

4.6.2.1 int _PXCContext::board_number

-1 to request any available

4.6.2.2 int _PXCContext::video_channel

Video channel number, see pxc.h.

4.6.2.3 int _PXCContext::pixel_format

see frame.h for image data types. Use PBITS_Y8 or PBITS_RGB24

4.6.2.4 int _PXCContext::video_format

see pxc.h for video detect types. Use NTSC_FORMAT

4.6.2.5 int _PXCContext::trigger_channel

-1 for no external triggering; else triggers on rising edge.

4.6.2.6 int _PXCContext::thread_priority

Priority of image capturing thread.

The documentation for this struct was generated from the following file:

- PXCCaptureLoop.hpp

4.7 _rgb Struct Reference

RGB pixel data type, 8 bits per channel (24 bpp). Use with class Pixmap.

`#include <Pixmap.hpp>`

Public Member Functions

- _rgb (uint8_t R=0, uint8_t G=0, uint8_t B=0)

Public Attributes

- uint8_t r

  red component

- uint8_t g

  green component

- uint8_t b

  blue component

4.7.1 Detailed Description

RGB pixel data type, 8 bits per channel (24 bpp). Use with class Pixmap.

For grayscale, the pixel data type is uint8_t.

4.7.2 Constructor & Destructor Documentation

4.7.2.1 _rgb::_rgb (uint8_t *R* = 0, uint8_t *G* = 0, uint8_t *B* = 0) `[inline]`

4.7.3 Member Data Documentation

4.7.3.1 uint8_t _rgb::r

red component

4.7.3.2 uint8_t _rgb::g

green component

4.7.3.3 uint8_t _rgb::b

blue component

The documentation for this struct was generated from the following file:

- Pixmap.hpp

4.8 CountFPS Class Reference

A frames-per-second counter.

`#include <TrackerUtils.hpp>`

Public Member Functions

- CountFPS ()
- ~CountFPS ()
- int init (int nFrames)
- void compute ()
- float report ()

4.8.1 Detailed Description

A frames-per-second counter.

This code was obtained from SDL webpage and encapsulated in a class. See: http://www.libsdl.org/cgi/docwiki.cgi/SDL_20Average_20FPS_20Measurement

4.8.2 Constructor & Destructor Documentation

4.8.2.1 CountFPS::CountFPS ()

Default constructor.

4.8.2.2 CountFPS::~CountFPS ()

Default destructor.

4.8.3 Member Function Documentation

4.8.3.1 int CountFPS::init (int *nFrames*)

Initialize counter. Call this method first before calling other methods.

Parameters:

nFrames The number of frames to average over in calculating the frame rate.

Returns:

0 on success, -1 on error.

4.8.3.2 void CountFPS::compute ()

Call this function every time a new frame is captured.

4.8.3.3 float CountFPS::report () `[inline]`

Report the current FPS calculation.

Returns:

Last computed average frame rate.

The documentation for this class was generated from the following file:

- TrackerUtils.hpp

4.9 FeatureClient Class Reference

A UDP network client for FeatureServer.

`#include <FeatureClientServer.hpp>`

Public Member Functions

- FeatureClient ()
- ~FeatureClient ()
- int initialize (const char *serverIp, int port, int msTimeOut, int nFeatures)
- int receiveFeatureList (feature_list_t &features)

4.9.1 Detailed Description

A UDP network client for FeatureServer.

Example Program: See examples for FeatureServer class.

4.9.2 Constructor & Destructor Documentation

4.9.2.1 FeatureClient::FeatureClient ()

Default constructor. Does nothing.

4.9.2.2 FeatureClient::~FeatureClient ()

Default destructor. Does nothing.

4.9.3 Member Function Documentation

4.9.3.1 int FeatureClient::initialize (const char * *serverIp*, int *port*, int *msTimeOut*, int *nFeatures*)

Connect to remote feature server.

Parameters:

serverIp The IP address of the remote server.

port The server port.

msTimeOut Connection timeout (in milliseconds).

nFeatures Number of feature points expected in the server message.

Returns:

0 on success, -1 on error (error message redirected to stderr).

4.9.3.2 int FeatureClient::receiveFeatureList (feature_list_t & *features*)

Parameters:

features feature list structure with updated features.

Returns:

0 on success, -1 on error (error message redirected to stderr).

The documentation for this class was generated from the following file:

- FeatureClientServer.hpp

4.10 FeatureServer Class Reference

A UDP network server for feature tracker.

`#include <FeatureClientServer.hpp>`

Public Member Functions

- FeatureServer ()
- ~FeatureServer ()
- int initialize (FeatureServerContext_t &cxt)
- int updateFeatures (feature_list_t &features, int srcFrameNumber)

Protected Member Functions

- virtual const char * receiveAndReply (const char *inMsgBuf, int inMsgLen, int *outMsgLen)
- virtual void enterThread (void *arg)
- virtual int executeInThread (void *arg)
- virtual void exitThread (void *arg)

4.10.1 Detailed Description

A UDP network server for feature tracker.

An object of this class starts a separate thread and replies to clients (FeatureClient object) with the latest feature point list.

Example Program:

```
//===============================================================================
// FeatureServer.t.cpp - Examples program for FeatureServer class
// UAV follower experiment
// Vilas Chitrakaran, May 2006
//===============================================================================

#include "FeatureClientServer.hpp"

int main()
{
 FeatureServer server;
 FeatureServerContext_t context;
 feature_list_t features;

 // set server parameters
 context.port = 8000;
 context.thread_priority = 10;
 context.num_features = 10;

 if( allocateFeatureList(features, context.num_features) < 0)
  return -1;

 // initialize and start server thread
 if( server.initialize(context) != 0)
  return -1;

 int frame = 0;
 while(1){
  // do processing here....
  // update server buffer
  features.features[0].x += 1;
  if( server.updateFeatures(features, frame++) != 0)
   break;
  sleep(1);
 }

 freeFeatureList(features);

 return 0;
}

//==============================================================================
// FeatureClient.t.cpp - Examples program for FeatureClient class
// UAV follower experiment
// Vilas Chitrakaran, May 2006
//==============================================================================

#include "FeatureClientServer.hpp"

//==============================================================================
// main: Connects to a feature server and delivers updates from server.
//==============================================================================
int main(int argc, char *argv[])
{
 FeatureClient client;
 feature_list_t features;
 int nFeatures = 10;

 // initialize a client and connect to server
 if(client.initialize("127.0.0.1", 8000, 50, nFeatures) != 0)
  return -1;

 // create feature list
 if( allocateFeatureList(features, nFeatures) < 0 )
  return -1;

 // server read loop
 int msgNum = 0;
 while(1) {
  if(msgNum > 1000) break;
  ++msgNum;

  // Ask for update from server
  if( client.receiveFeatureList(features) == -1)
   break;

  if( features.frame_number < 0 ) // server hasn't started updating frames yet
   continue;

  // print recevied frame number
  fprintf(stdout, "latest frame received: %d.\n", features.frame_number);
 }

 freeFeatureList(features);

 return 0;
}
```

4.10.2 Constructor & Destructor Documentation

4.10.2.1 FeatureServer::FeatureServer ()

Default constructor. Does a few initializations.

4.10.2.2 FeatureServer::∼FeatureServer ()

Default destructor. Frees resources

4.10.3 Member Function Documentation

4.10.3.1 int FeatureServer::initialize (FeatureServerContext_t & *cxt*)

Initializes and starts the server thread. This must be the first method called before using any other method in this class.

Returns:

0 on success, -1 on error (error message redirected to stderr).

4.10.3.2 int FeatureServer::updateFeatures (feature_list_t & *features*, int *srcFrameNumber*)

Update the features buffer in the server.

Parameters:

features The feature list.

srcFrameNumber the image/video frame number corresponding to this feature list. The internal buffer is not updated unless this number is different from an internally maintained counter. This avoid unecessary copy operations.

Returns:

0 on success, -1 on error (error message redirected to stderr).

4.10.3.3 virtual const char∗ FeatureServer::receiveAndReply (const char ∗ *inMsgBuf*, int *inMsgLen*, int ∗ *outMsgLen*) `[protected, virtual]`

Reimplemented from UDPServer class.

4.10.3.4 virtual void FeatureServer::enterThread (void ∗ *arg*) `[protected, virtual]`

Reimplemented from Thread class.

4.10.3.5 virtual int FeatureServer::executeInThread (void ∗ *arg*) `[protected, virtual]`

Reimplemented from Thread class.

4.10.3.6 virtual void FeatureServer::exitThread (void ∗ *arg*) `[protected, virtual]`

Reimplemented from Thread class.

The documentation for this class was generated from the following file:

- FeatureClientServer.hpp

4.11 FeatureTrackerKLT Class Reference

Automatic image feature detection and tracking using the Lucas-Kanade tracking algorithm implemented in the KLT library.

`#include <FeatureTrackerKLT.hpp>`

Public Member Functions

- FeatureTrackerKLT ()
- ∼FeatureTrackerKLT ()
- int initialize (FeatureTrackerContext_t &ftc, KLT_TrackingContext kltc)
- int processImage (unsigned char ∗img, int w, int h, feature_list_t &list)
- int writeFeatureTable (const char ∗fileBaseName)

4.11.1 Detailed Description

Automatic image feature detection and tracking using the Lucas-Kanade tracking algorithm implemented in the KLT library.

This class uses the implementation of the KLT algorithm developed and maintained by Stan Birchfield. See: http://www.ces.clemson.edu/∼stb/klt . Image display and event handling routines use the SDL library . See: http://www.libsdl.org.

Example Program:

```
//==============================================================================
// FeatureTrackerKLT.t.cpp - Example program for FeatureTrackerKLT class
// Vilas Chitrakaran, May 2006
//==============================================================================

#include "FeatureTrackerKLT.hpp"
#include "Pixmap.hpp"
#include <unistd.h>

int main()
{
 KLT_TrackingContext kltContext;
 FeatureTrackerContext_t tracContext;
 PixmapGray img[2];
 feature_list_t features;

 kltContext = KLTCreateTrackingContext();
 kltContext->lighting_insensitive = true;
 kltContext->writeInternalImages = false;
 kltContext->affineConsistencyCheck = -1;
 kltContext->window_width = 9;
 kltContext->window_height = 9;
 kltContext->max_iterations = 100;
 kltContext->mindist = 10;
 kltContext->smoothBeforeSelecting = true;

 tracContext.num_features = 4;
 tracContext.num_frames = 2;
 tracContext.auto_select_features = false;
 tracContext.display_tracked_features = true;

 // create feature list
 if( allocateFeatureList(features, tracContext.num_features) < 0 )
  return -1;
 FeatureTrackerKLT tracker;

 img[0].loadPixmap("images/box0.pgm");
 img[1].loadPixmap("images/box1.pgm");

 // initialize system
 if( tracker.initialize(tracContext, kltContext) != 0 ) {
  fprintf(stderr, "ERROR initializing tracker.\n");
  return -1;
 }

 // track features between frames
 for(int i = 0; i < 2; ++i) {
  if( tracker.processImage(img[i].getPointer(0), img[i].getWidth(),
                           img[i].getHeight(), features) < 0 ) {
   fprintf(stderr, "ERROR processing image.\n");
   return -1;
  }

  // print features
  fprintf(stdout, "== frame %2d ==\n", i);
  for(int j = 0; j < tracContext.num_features; ++j) {
   fprintf(stdout, "%2d (%3.1f, %3.1f)\n", features.features[j].val,
           features.features[j].x, features.features[j].y);
  }
 }

 // SDL events won't be caught outside processImage(), unless
 // you do this...
 SDL_Event event;
 while( SDL_PollEvent(&event)  ) {
  if(event.type == SDL_QUIT) {
   fprintf(stdout, "\nI was asked to quit!\n");
   return -2;
  }
 }

 // Write results to text.
 if( tracker.writeFeatureTable("trackedFeatures") != 0 ) {
  fprintf(stderr, "ERROR writing feature tables.\n");
  return -1;
 }

 freeFeatureList(features);
 KLTFreeTrackingContext(kltContext);

 return 0;
}
```

4.11.2 Constructor & Destructor Documentation

4.11.2.1 FeatureTrackerKLT::FeatureTrackerKLT ()

Default constructor.

4.11.2.2 FeatureTrackerKLT::~FeatureTrackerKLT ()

Destructor frees any allocated resources

4.11.3 Member Function Documentation

4.11.3.1 int FeatureTrackerKLT::initialize (FeatureTrackerContext_t & *ftc*, KLT_TrackingContext *kltc*)

Initialize the tracker. Call this method before calling any other methods of this class.

Parameters:

ftc settings specific to this class.

kltc KLT algorithm specific settings.

Returns:

0 on success, -1 on error (error message redirected to stderr).

4.11.3.2 int FeatureTrackerKLT::processImage (unsigned char ∗ *img*, int *w*, int *h*, feature_list_t & *list*)

Track features in the image buffer. Upon calling this method the first time, features are selected either automatically (if 'auto_select_features' was turned on during initialization) or by the user. If 'display_tracked_features' was turned on during initialization, the image display window will be updated with the current image in the buffer and the location of tracked features are marked.

NOTE: Calling this function initiates SDL event handling, including for SIGINT (CNTRL+C). Hence, to catch events outside of this method, use SDL functions such as SDL_PollEvent().

Parameters:

img Pointer to image buffer. NOTE: image must be 8 bit grayscale.

w,h Image dimensions in pixels.

list List of tracked features. This list contains updated (x,y) locations of features and an integer value indicating whether the feature was tracked successfully (0) or not (-1).

Returns:

current frame number on success (first frame = 1), -1 on error (error message redirected to stderr), -2 on user initiated quit.

4.11.3.3 int FeatureTrackerKLT::writeFeatureTable (const char ∗ *fileBaseName*)

Write the history of all tracked features into a feature table in ascii (.txt) and binary format (.ft). (See KLT library documentation for details on reading from feature table). The number of records in this feature table will be less than or equal to the 'num_frames' parameter in the FeatureTrackerContext_t passed to initialize().

Parameters:

fileBaseName Base name of the file.

Returns:

0 on success, -1 on error (error message redirected to stderr).

The documentation for this class was generated from the following file:

- FeatureTrackerKLT.hpp

4.12 FeatureTrackerOCV Class Reference

Automatic image feature detection and tracking using the OpenCV library implementation of the Lucas-Kanade tracking algorithm.

`#include <FeatureTrackerOCV.hpp>`

Public Member Functions

- FeatureTrackerOCV ()
- ~FeatureTrackerOCV ()
- int initialize (FeatureTrackerContext_t &ftc, OCVTrackingContext_t &ocvt)
- int processImage (unsigned char *img, int w, int h, feature_list_t &list)

4.12.1 Detailed Description

Automatic image feature detection and tracking using the OpenCV library implementation of the Lucas-Kanade tracking algorithm.

OpenCV must be installed in order to use this class. See: http://www.intel.com/technology/computing/opencv/index.htm .

Image display and event handling routines use the SDL library. See: http://www.libsdl.org .

Example Program:

```
//==============================================================================
// FeatureTrackerKLT.t.cpp - Example program for FeatureTrackerOCV class
// Vilas Chitrakaran, May 2006
//==============================================================================

#include "FeatureTrackerOCV.hpp"
#include "Pixmap.hpp"
#include <unistd.h>

int main()
{
 OCVTrackingContext_t ocvContext;
 FeatureTrackerContext_t tracContext;
 PixmapGray img[2];
 feature_list_t features;

 ocvContext.min_dist = 20;
 ocvContext.quality = 0.001;
 ocvContext.block_size = 5;
 ocvContext.max_iter = 100;
 ocvContext.epsilon = 0.01;
 ocvContext.window_size = 3;
 ocvContext.max_error = 300;

 tracContext.num_features = 4;
 tracContext.num_frames = 2;
 tracContext.auto_select_features = false;
 tracContext.display_tracked_features = true;

 // create feature list
 if( allocateFeatureList(features, tracContext.num_features) < 0 )
  return -1;

 FeatureTrackerOCV tracker;
 img[0].loadPixmap("images/box0.pgm");
 img[1].loadPixmap("images/box1.pgm");

 // initialize system
 if( tracker.initialize(tracContext, ocvContext) != 0 ) {
  fprintf(stderr, "ERROR initializing tracker.\n");
  return -1;
 }

 // track features between frames
 for(int i = 0; i < 2; ++i) {
  if( tracker.processImage(img[i].getPointer(0), img[i].getWidth(),
                           img[i].getHeight(), features) < 0 ) {
   fprintf(stderr, "ERROR processing image.\n");
   return -1;
  }

  // print features
  fprintf(stdout, "== frame %2d ==\n", i);
  for(int j = 0; j < tracContext.num_features; ++j) {
   fprintf(stdout, "%2d (%3.1f, %3.1f)\n", features.features[j].val,
           features.features[j].x, features.features[j].y);
  }
 }

 // SDL events won't be caught outside processImage(), unless
 // you do this...
 SDL_Event event;
 while( SDL_PollEvent(&event)  ) {
  if(event.type == SDL_QUIT) {
   fprintf(stdout, "\nI was asked to quit!\n");
   return -2;
  }
 }

 freeFeatureList(features);

 return 0;
}
```

4.12.2 Constructor & Destructor Documentation

4.12.2.1 FeatureTrackerOCV::FeatureTrackerOCV ()

Default constructor.

4.12.2.2 FeatureTrackerOCV::~FeatureTrackerOCV ()

Destructor frees any allocated resources.

4.12.3 Member Function Documentation

4.12.3.1 int FeatureTrackerOCV::initialize (FeatureTrackerContext_t & *ftc*, OCVTrackingContext_t & *ocvt*)

Initialize the tracker. Call this method before calling any other methods of this class.

Parameters:

ftc settings specific to this class.

ocvt tracker algorithm specific settings.

Returns:

0 on success, -1 on error (error message redirected to stderr).

4.12.3.2 int FeatureTrackerOCV::processImage (unsigned char * *img*, int *w*, int *h*, feature_list_t & *list*)

Track features in the image buffer. Upon calling this method the first time, features are selected either automatically (if 'auto_select_features' was turned on during initialization) or by the user. If 'display_tracked_features' was turned on during initialization, the image display window will be updated with the current image in the buffer and the location of tracked features are marked.

NOTE: Calling this function initiates SDL event handling, including for SIGINT (CNTRL+C). Hence, to catch events outside of this method, use SDL functions such as SDL_PollEvent().

Parameters:

img Pointer to image buffer. NOTE: image must be 8 bit grayscale.

w,h Image dimensions in pixels.

list List of tracked features. This list contains updated (x,y) locations of features and an integer value indicating whether the feature was tracked successfully (0) or not (-1).

Returns:

current frame number on success (first frame = 1), -1 on error (error message redirected to stderr), -2 on user initiated quit.

The documentation for this class was generated from the following file:

- FeatureTrackerOCV.hpp

4.13 Pixmap< T > Class Template Reference

The template class for pixmap (ppm, pgm) images.

`#include <Pixmap.hpp>`

Public Member Functions

- Pixmap ()
- Pixmap (int w, int h)
- Pixmap (uint8_t *buffer, int w, int h)
- virtual ~Pixmap ()
- int create (int w, int h)
- int attach (uint8_t *buffer, int w, int h)
- int getWidth () const
- int getHeight () const
- int getBytesPerPixel () const
- bool isIndexValid (int i)
- bool isIndexValid (int c, int r)
- T * getPointer (int i)
- T * getPointer (int c, int r)
- T & operator() (int i)
- T & operator() (int c, int r)
- Pixmap< T > & operator= (const Pixmap< T > &p)
- int loadPixmap (const char *fileName)
- int savePixmap (char *fileName)

Protected Attributes

- bool d_usingExternalBuffer
- T * d_imgData
- int d_w
- int d_h

4.13.1 Detailed Description

template<class T> class Pixmap< T >

The template class for pixmap (ppm, pgm) images.

The class provides methods to read and write images only as pixmaps (PPM). However, methods to directly access the image buffer is provided, hence the user can develop her own additional functions to support other image formats. The image buffer created by this class stores images either as 8 bit grayscale (T = uint8_t), or 24 bit RGB (T = rgb_t) in packed pixel format (ie, all the data for a pixel lie next to each other in memory.

Example Program:

```
//==============================================================================
// Pixmap.t.cpp : Example program for Pixmap class.
// Author       : Vilas Kumar Chitrakaran
//==============================================================================
#include "Pixmap.hpp"

//==============================================================================
// This example demonstrates how to read an image, modify it and
// write it back as a file.
//==============================================================================
using namespace std;

int main()
{
 PixmapRgb img;

 // open an image
 if( img.loadPixmap("images/ash_P6.ppm") != 0 ) {
  fprintf(stderr, "OOPS\n");
  return -1;
 }

 // print image dimensions.
 fprintf(stdout, "Opened image of size: %d x %d\n",
         img.getWidth(), img.getHeight() );

 // modify a pixel
 img(3,4) = rgb_t(255,0,0);

 // save to file
 if( img.savePixmap("new_image.ppm") != 0 ) {
  fprintf(stderr, "OOPS\n");
  return -1;
 }

 return 0;
}
```

4.13.2 Constructor & Destructor Documentation

4.13.2.1 template<class T> Pixmap< T >::Pixmap ()

Default constructor. Does nothing

4.13.2.2 template<class T> Pixmap< T >::Pixmap (int *w*, int *h*)

Constructor that allocates memory buffer.

Parameters:

w image width (pixels)

h image height (pixels)

4.13.2.3 template<class T> Pixmap< T >::Pixmap (uint8_t * *buffer*, int *w*, int *h*)

Constructor that hooks to an externally allocated memory buffer instead of allocating memory of it's own. It is user's responsbility to ensure that buffer size is adequate for an image of size w x h of specified type.

Parameters:

buffer Pointer to image data

w image width (pixels)

h image height (pixels)

4.13.2.4 template<class T> virtual Pixmap< T >::~Pixmap () [virtual]

The destructor. Frees any allocated memory.

4.13.3 Member Function Documentation

4.13.3.1 template<class T> int Pixmap< T >::create (int *w*, int *h*)

Allocates a new data buffer for image data, or resizes a previously allocated buffer. The buffer values are not initialized, and may be anything arbitrary.

Parameters:

w width (pixels).

h height (pixels).

Returns:

0 on success, -1 if failed.

4.13.3.2 template<class T> int Pixmap< T >::attach (uint8_t ∗ *buffer*, int *w*, int *h*)

Hook to an externally provided buffer for image data (such as framebuffer of a frame grabber). It is user's responsbility to ensure that buffer size is adequate for an image of size w x h of specified type.

Parameters:

buffer Pointer to image data

w width (pixels).

h height (pixels).

Returns:

0 on success, -1 if failed.

4.13.3.3 template<class T> int Pixmap< T >::getWidth () const [inline]

Returns:

width of image in pixels.

4.13.3.4 template<class T> int Pixmap< T >::getHeight () const [inline]

Returns:

height of image in pixels.

4.13.3.5 template<class T> int Pixmap< T >::getBytesPerPixel () const [inline]

Returns:

Bytes per pixel.

4.13.3.6 template<class T> bool Pixmap< T >::isIndexValid (int *i*) [inline]

Parameters:

i 1D index into data buffer (starts at 0).

Returns:

true if index within image boundaries, else false.

4.13.3.7 template<class T> bool Pixmap< T >::isIndexValid (int *c*, int *r*) [inline]

Parameters:

c*,*r column, row index into data buffer (starts at (0,0)).

Returns:

true if index within image boundaries, else false.

4.13.3.8 template<class T> T* Pixmap< T >::getPointer (int *i*) [inline]

Parameters:

i 1D index into data buffer (starts at 0).

Returns:

Pointer to pixel data at specified index, NULL if index is out of range.

4.13.3.9 template<class T> T* Pixmap< T >::getPointer (int *c*, int *r*) [inline]

Parameters:

c*,*r column, row index into data buffer (starts at (0,0)).

Returns:

Pointer to pixel data at specified index, NULL if index is out of range.

4.13.3.10 template<class T> T& Pixmap< T >::operator() (int *i*) [inline]

Access image data at a specified location. For example:

```
myImage(2)=255;
```

or

```
cout << myImage(2) << endl;
```

Parameters:

i 1D index into data buffer (starts at 0).

Returns:

Pointer to pixel data at specified index

4.13.3.11 template<class T> T& Pixmap< T >::operator() (int *c*, int *r*) [inline]

Access image data at a specified location. For example:

```
myImage(2,2)=255;
```

or

```
cout << myImage(2,2) << endl;
```

Parameters:

c*,*r column, row index into data buffer (starts at (0,0)).

Returns:

Pointer to pixel data at specified index

4.13.3.12 template<class T> Pixmap<T>& Pixmap< T >::operator= (const Pixmap< T > & *p*)

Assignment between two images of same type and dimensions.

Parameters:

p The Pixmap object.

4.13.3.13 template<class T> int Pixmap< T >::loadPixmap (const char ∗ *fileName*)

Load a pixmap image (pgm, ppm).

Parameters:

fileName The name of the image file

Returns:

0 on success, -1 on error.

4.13.3.14 template<class T> int Pixmap< T >::savePixmap (char ∗ *fileName*)

Save the image as a pixmap (ppm)

Parameters:

fileName The name of the image file

Returns:

0 on success, -1 on error.

4.13.4 Member Data Documentation

4.13.4.1 template<class T> bool Pixmap< T >::d_usingExternalBuffer [protected]

4.13.4.2 template<class T> T* Pixmap< T >::d_imgData [protected]

4.13.4.3 template<class T> int Pixmap< T >::d_w [protected]

4.13.4.4 template<class T> int Pixmap< T >::d_h [protected]

The documentation for this class was generated from the following file:

- Pixmap.hpp

4.14 PixmapGray Class Reference

Class for 1 byte-per-pixel greyscale (pgm) images.

`#include <Pixmap.hpp>`

Inheritance diagram for PixmapGray::

4.14.1 Detailed Description

Class for 1 byte-per-pixel greyscale (pgm) images.

The documentation for this class was generated from the following file:

- Pixmap.hpp

4.15 PixmapRgb Class Reference

Class for 3 bytes-per-pixel RGB (ppm) images.

`#include <Pixmap.hpp>`

Inheritance diagram for PixmapRgb::

4.15.1 Detailed Description

Class for 3 bytes-per-pixel RGB (ppm) images.

The documentation for this class was generated from the following file:

- Pixmap.hpp

4.16 PXCCaptureLoop Class Reference

A QNX specific interface to capture images using the Imagenation PXC200AF frame grabber.

`#include <PXCCaptureLoop.hpp>`

Public Member Functions

- PXCCaptureLoop ()
- virtual ∼PXCCaptureLoop ()
- int initialize (PXCContext_t &cxt)
- int startCaptureLoop ()
- int getImageProperties (int &w, int &h, int &bpp)

Protected Member Functions

- virtual int processImage (const unsigned char ∗fbr, int w, int h, int bpp)
- virtual void enterThread (void ∗arg)
- virtual int executeInThread (void ∗arg)
- virtual void exitThread (void ∗arg)

Protected Attributes

- int d_imgWidth
- int d_imgHeight
- int d_bpp

4.16.1 Detailed Description

A QNX specific interface to capture images using the Imagenation PXC200AF frame grabber.

An object of this class interfaces with a PXC200 AF framegrabber through its device driver. The object initiates a separate thread for image capturing and transfers image data to a user specified memory buffer through a user implemented function. This class is specifically written to work with the QNX 6.2.1 device driver for PXC200AF. More information on PXC series framegrabbers are available here: http://www.imagenation.com/pxcfamily.html.

Example Program:

```
//============================================================================
// PXCCaptureLoop.t.cpp - Examples program for PXCCaptureLoop class
// UAV follower experiment
// Vilas Chitrakaran, May 2006
//============================================================================

#include "PXCCaptureLoop.hpp"


int main()
{
 PXCContext_t settings;
 PXCCaptureLoop camera;

 settings.board_number = -1;
 settings.video_channel = 0;
 settings.pixel_format = PBITS_Y8;
 settings.video_format = NTSC_FORMAT;
 settings.trigger_channel = -1;
 settings.thread_priority = 10;

 if( camera.initialize(settings) != 0 )
  return -1;

 if( camera.startCaptureLoop() != 0 )
  return -1;

 fprintf(stdout, "thread started\n");
 sleep(10);

 return 0;
}
```

4.16.2 Constructor & Destructor Documentation

4.16.2.1 PXCCaptureLoop::PXCCaptureLoop ()

Default constructor.

4.16.2.2 virtual PXCCaptureLoop::~PXCCaptureLoop () [virtual]

Destructor. Frees any allocated resources, shuts down the framegrabber.

4.16.3 Member Function Documentation

4.16.3.1 int PXCCaptureLoop::initialize (PXCContext_t & *cxt*)

Initializes the frame grabber.

Parameters:

cxt settings specific to framegrabber.

Returns:

0 on success, -1 on error (error message redirected to stderr).

4.16.3.2 int PXCCaptureLoop::startCaptureLoop ()

start image capture loop. Method processImage() is called everytime a new image is acquired.

Returns:

0 on success, -1 on error (error message redirected to stderr).

4.16.3.3 int PXCCaptureLoop::getImageProperties (int & *w*, int & *h*, int & *bpp*)

Get properties of the images being captured by the camera

Parameters:

w,h Width and height of the image.

bpp Image bytes per pixel (1 = 8 bit grayscale, 3 = 24 bit RGB).

Returns:

0 on success, -1 on error (error message redirected to stderr).

4.16.3.4 virtual int PXCCaptureLoop::processImage (const unsigned char ∗ *fbr*, int *w*, int *h*, int *bpp*) `[protected, virtual]`

Reimplement this function in a derived class. This method is automatically called upon every successful acquisition of an image. It is upto the implementation to ensure that processing delays do not cause dropped frames. A suggested implementation would do nothing more than memcpy() the framebuffer to a user specified buffer and return immediately. A separate thread/process can then process the contents of the copied buffer.

Parameters:

fbr Pointer to frame buffer containing current image update.

w,h Width and height of the image.

bpp Image bytes per pixel (1 = 8 bit grayscale, 3 = 24 bit RGB).

Returns:

Implementation must return 0 on success, non-zero on error.

4.16.3.5 virtual void PXCCaptureLoop::enterThread (void ∗ *arg*) `[protected, virtual]`

Reimplemented from Thread class

4.16.3.6 virtual int PXCCaptureLoop::executeInThread (void ∗ *arg*) `[protected, virtual]`

Reimplemented from Thread class

4.16.3.7 virtual void PXCCaptureLoop::exitThread (void ∗ *arg*) `[protected, virtual]`

Reimplemented from Thread class

4.16.4 Member Data Documentation

4.16.4.1 int PXCCaptureLoop::d_imgWidth `[protected]`

4.16.4.2 int PXCCaptureLoop::d_imgHeight `[protected]`

4.16.4.3 int PXCCaptureLoop::d_bpp `[protected]`

The documentation for this class was generated from the following file:

- PXCCaptureLoop.hpp

4.17 SDLWindow Class Reference

A window for image display.

`#include <TrackerUtils.hpp>`

Public Member Functions

- SDLWindow ()
- ~SDLWindow ()
- int init (int w, int h, const char ∗title)
- int updateScreenBuffer (char ∗buf, int w, int h, int bpp, const char ∗msg=NULL)
- void refresh ()
- SDL_Surface ∗ getScreenPointer ()

4.17.1 Detailed Description

A window for image display.

The SDLWindow class uses the SDL library to display images. Use SDL event handling routines to catch events such as mouse clicks.

Example Program:

```
//==============================================================================
// SDLWindow.t.cpp : Example program for SDLWindow and Pixmap class.
// Author          : Vilas Kumar Chitrakaran
//==============================================================================

#include "TrackerUtils.hpp"
#include "SDL/SDL_events.h"
#include "Pixmap.hpp"
#include <iostream>

using namespace std;

static int quit = 0;
int filterSDLQuitEvent(const SDL_Event *event);
 // filter out SDL_QUIT and handle it here.

//==============================================================================
// This example demonstrates how to display an image, and process user mouse
// clicks.
//==============================================================================
int main(int argc, char *argv[])
{
 SDLWindow window;  // image window
 PixmapRgb image;   // image
 char *pointer;
 int w, h, bpp;

 // open an image
 if( image.loadPixmap("images/ash_P6.ppm") != 0 )
  return -1;

 pointer = (char *)image.getPointer(0);
 w = image.getWidth();
 h = image.getHeight();
 bpp = image.getBytesPerPixel();

 // Display the image on screen
 if( window.updateScreenBuffer(pointer, w, h, bpp, NULL) != 0)
  return -1;
 window.refresh();

 // handle mouse events (standard SDL event handling)
 SDL_Event event;
 SDL_SetEventFilter(filterSDLQuitEvent); // handle SDL_QUIT
 while( SDL_WaitEvent(&event) && !quit ) {
  switch(event.type) {
   int x, y;
   case SDL_MOUSEBUTTONDOWN:
    x = event.button.x;
    y = event.button.y;
    cout << image(x,y) << endl;
    image(x,y) = rgb_t(0xFF, 0, 0);
   break;

   default:
   break;
  }
  if( window.updateScreenBuffer(pointer, w, h, bpp, NULL) != 0)
   return -1;
  window.refresh();
 }
 return 0;
}


//=============================================================================
// filterSDLQuitEvent
//=============================================================================
int filterSDLQuitEvent(const SDL_Event *event)
{
 if( event->type == SDL_QUIT) {
  cout << "Quitting." << endl;
  quit = 1;
 }
 return(1);
}
```

4.17.2 Constructor & Destructor Documentation

4.17.2.1 SDLWindow::SDLWindow ()

The default constructor. Does some initializations.

4.17.2.2 SDLWindow::∼SDLWindow ()

The destructor cleans up.

4.17.3 Member Function Documentation

4.17.3.1 int SDLWindow::init (int *w*, int *h*, const char ∗ *title*)

Initialize an SDL screen buffer. Window doesn't show up until refresh() is called.

Parameters:

w,h Window width and height.

title A title for the window. Should be set to NULL if not desired.

Returns:

0 on success, -1 on error (error message redirected to stderr).

4.17.3.2 int SDLWindow::updateScreenBuffer (char ∗ *buf*, int *w*, int *h*, int *bpp*, const char ∗ *msg* = NULL)

Update the screen buffer with data from user provided image buffer. Window doesn't show up on until refresh() is called.

Parameters:

buf A pointer to the image buffer, provided by the user.

w,h Image width and height. Provided here to reinitialize SDL screen if they are different from parameters used for init().

bpp The bytes per pixel.

msg An optional message upto 80 characters long. Useful to print helpful information on the screen.

Returns:

0 on success, -1 on error (error message redirected to stderr).

4.17.3.3 void SDLWindow::refresh ()

Display the video on screen.

4.17.3.4 SDL_Surface∗ SDLWindow::getScreenPointer () `[inline]`

The documentation for this class was generated from the following file:

- TrackerUtils.hpp

Chapter 5

Real-time Feature Tracker Library File Documentation

5.1 FeatureClientServer.hpp File Reference

```
#include "putils/UDPClientServer.hpp"
#include "putils/RWLock.hpp"
#include "putils/Thread.hpp"
#include "TrackerUtils.hpp"
```

Classes

- struct _FeatureServerContext

  Parameters for UDP feature server. Use with class FeatureServer.

- class FeatureServer

  A UDP network server for feature tracker.

- class FeatureClient

  A UDP network client for FeatureServer.

Typedefs

- typedef _FeatureServerContext FeatureServerContext_t

5.1.1 Typedef Documentation

5.1.1.1 typedef struct _FeatureServerContext FeatureServerContext_t

5.2 FeatureTrackerKLT.hpp File Reference

```
#include "klt/klt.h"
#include <stdio.h>
#include <malloc.h>
#include "TrackerUtils.hpp"
```

Classes

- class FeatureTrackerKLT

  Automatic image feature detection and tracking using the Lucas-Kanade tracking algorithm implemented in the KLT library.

5.3 FeatureTrackerOCV.hpp File Reference

```
#include "opencv/cv.h"
#include <stdio.h>
#include <malloc.h>
#include "TrackerUtils.hpp"
```

Classes

- struct _OCVTrackingContext

  Parameters specific to OpenCV tracker (for use with FeatureTrackerOCV class).

- class FeatureTrackerOCV

  Automatic image feature detection and tracking using the OpenCV library implementation of the Lucas-Kanade tracking algorithm.

Typedefs

- typedef _OCVTrackingContext OCVTrackingContext_t

5.3.1 Typedef Documentation

5.3.1.1 typedef struct _OCVTrackingContext OCVTrackingContext_t

5.4 Pixmap.hpp File Reference

```
#include <math.h>
#include <malloc.h>
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <typeinfo>
#include <inttypes.h>
```

Classes

- struct _rgb

  RGB pixel data type, 8 bits per channel (24 bpp). Use with class Pixmap.

- class Pixmap< T >

  The template class for pixmap (ppm, pgm) images.

- class PixmapGray

  Class for 1 byte-per-pixel greyscale (pgm) images.

- class PixmapRgb

  Class for 3 bytes-per-pixel RGB (ppm) images.

Typedefs

- typedef _rgb rgb_t

Functions

- bool operator== (const rgb_t &c1, const rgb_t &c2)
- bool operator!= (const rgb_t &c1, const rgb_t &c2)
- std::ostream & operator<< (std::ostream &out, const rgb_t &rgb)
- std::istream & operator>> (std::istream &in, rgb_t &rgb)

5.4.1 Typedef Documentation

5.4.1.1 typedef struct _rgb rgb_t

5.4.2 Function Documentation

5.4.2.1 bool operator== (const rgb_t & *c1*, const rgb_t & *c2*)

5.4.2.2 bool operator!= (const rgb_t & *c1*, const rgb_t & *c2*)

5.4.2.3 std::ostream& operator<< (std::ostream & *out*, const rgb_t & *rgb*)

5.4.2.4 std::istream& operator>> (std::istream & *in*, rgb_t & *rgb*)

5.5 PXCCaptureLoop.hpp File Reference

```
#include "pxc200/pxc.h"
#include "pxc200/frame.h"
#include "putils/Thread.hpp"
#include <stdio.h>
```

Classes

- struct _PXCContext

  Parameters for the PXC200AF framegrabber. Use with class PXCCaptureLoop.

- class PXCCaptureLoop

  A QNX specific interface to capture images using the Imagenation PXC200AF frame grabber.

Typedefs

- typedef _PXCContext PXCContext_t

5.5.1 Typedef Documentation

5.5.1.1 typedef struct _PXCContext PXCContext_t

5.6 TrackerUtils.hpp File Reference

```
#include "SDL/SDL.h"
#include "SDL/SDL_gfxPrimitives.h"
#include "klt/klt.h"
```

Classes

- struct _FeatureTrackerContext

  Parameters for feature tracking.
- struct _feature

  A feature point.
- struct _feature_list

  List of features in a single image.
- class CountFPS

  A frames-per-second counter.
- class SDLWindow

  A window for image display.

Typedefs

- typedef _FeatureTrackerContext FeatureTrackerContext_t
- typedef _feature feature_t
- typedef _feature_list feature_list_t

Functions

- int allocateFeatureList (feature_list_t &f, int num_features)
- void freeFeatureList (feature_list_t &f)
- int copyFeaturesToKLTFeatureList (feature_list_t &f, KLT_FeatureList kl)

5.6.1 Typedef Documentation

5.6.1.1 typedef struct _FeatureTrackerContext FeatureTrackerContext_t

5.6.1.2 typedef struct _feature feature_t

5.6.1.3 typedef struct _feature_list feature_list_t

5.6.2 Function Documentation

5.6.2.1 int allocateFeatureList (feature_list_t & *f*, int *num_features*)

Allocate memory for storing features.

Returns:

Size (bytes) of the entire buffer on success, -1 on error;

5.6.2.2 void freeFeatureList (feature_list_t & *f*)

Free the memory allocated for storing features using allocateFeatureStruct().

5.6.2.3 int copyFeaturesToKLTFeatureList (feature_list_t & *f*, KLT_FeatureList *kl*)

Copy a feature list into feature list structure used in the KLT library.

Returns:

0 on success, -1 on error (error message redirected to stderr).

Index