Orientation Estimation

from ceiling-mounted cameras

Code documentation – Master's Thesis in Artificial Intelligence (Intelligent Systems)

Silvia-Laura Pintea *(6109969)*

<S.L.Pintea@student.uva.nl>

Contents

Chapter 1

Data Structure Documentation

1.1 AnnotationsHandle::ANNOTATION Struct Reference

A structure that stores a single annotation for a specific person.

Public Member Functions

- ANNOTATION (const ANNOTATION &anno)
- ANNOTATION & **operator=** (const ANNOTATION &anno)

Data Fields

- short int **id_**
- cv::Point2f **location_**
- std::deque< unsigned int > **poses_**

1.2 AnnotationsHandle Class Reference

Class for annotating both positions and poses of the people in the images.

Data Structures

- struct ANNOTATION

  A structure that stores a single annotation for a specific person.

- struct ASSIGNED

  Shows which id from the old annotations is assigned to which id from the new annotations based on what minimal distance.

- struct FULL_ANNOTATIONS

  Structure containing a vector of annotations for each image.

Public Types

- enum POSE {
  SITTING, STANDING, BENDING, LONGITUDE,
  LATITUDE }

  All considered poses.

Static Public Member Functions

- static void mouseHandlerAnn (int event, int x, int y, int flags, void ∗param)

  Mouse handler for annotating people's positions and poses.

- static void showMenu (const cv::Point2f &center)

  Draws the "menu" of possible poses for the current position.

- static void plotHull (IplImage ∗img, std::vector< cv::Point2f > &hull)

  Plots the hull indicated by the parameter `hull` *on the given image.*

- static int runAnn (int argc, char ∗∗argv, unsigned step, const std::string &usedImages, int imgIndex=-1)

  Starts the annotation process for the images.

- static void trackbar_callback (int position, void ∗param)

  The "on change" handler for the track-bars.

- static void trackBarHandleFct (int position, void ∗param)

  A function that starts a new thread which handles the track-bar event.

- static void loadAnnotations (char ∗filename, std::deque< AnnotationsHandle::FULL_ANNOTATIONS > &loadedAnno)

  Load annotations from file.

- static void annoDifferences (std::deque< AnnotationsHandle::FULL_ANNOTATIONS > &train, std::deque< AnnotationsHandle::FULL_ANNOTATIONS > &test, float &avgDist, float &Ndiff, float ssdLongDiff, float ssdLatDiff, float poseDiff)

  Computes the average distance from the predicted location and the annotated one,the number of unpredicted people in each image and the differences in the pose estimation.

- static void correltateLocs (std::deque< AnnotationsHandle::ANNOTATION > &annoOld, std::deque< AnnotationsHandle::ANNOTATION > &annoNew, std::deque< AnnotationsHandle::ASSIGNED > &idAssignedTo)

  Correlate annotations' from locations in `annoOld` *to locations in* `annoNew` *through IDs.*

- static bool canBeAssigned (std::deque< AnnotationsHandle::ASSIGNED > &idAssignedTo, short int id, float newDist, short int to)

  Checks to see if a location can be assigned to a specific ID given the new distance.

- static void displayFullAnns (std::deque< AnnotationsHandle::FULL_ANNOTATIONS > &fullAnns)

  Displays the complete annotations for all images.

- static int runEvaluation (int argc, char **argv)

  Evaluates the annotation of the images.

- static void drawOrientation (const cv::Point2f &center, unsigned int orient, const std::tr1::shared_ptr< IplImage > im)

  Shows how the selected orientation looks on the image.

- static cv::Mat drawOrientation (const cv::Point2f &center, unsigned int orient, const cv::Mat &im, const cv::Scalar &color)

  Overloaded version for cv::Mat -- shows how the selected orientation looks on the image.

- static void drawLatitude (const cv::Point2f &head, const cv::Point2f &feet, unsigned int orient, AnnotationsHandle::POSE pose)

  Shows how the selected orientation looks on the image.

- static cv::Mat rotateWrtCamera (const cv::Point2f &headLocation, const cv::Point2f &feetLocation, const cv::Mat &toRotate, cv::Point2f &borders)

  Rotate matrix wrt to the camera location.

- static void writeAnnoToFile (const std::deque< AnnotationsHandle::FULL_ANNOTATIONS > &fullAnno, const std::string &fileName)

  Writes a given FULL_ANNOTATIONS structure into a given file.

- static void init ()

  Initializes all the values of the class variables.

- static void checkCalibration (int argc, char **argv)

  Check calibration: shows how the projection grows depending on the location of the point.

- static int runAnnArtificial (int argc, char **argv, unsigned step, const std::string &usedImages, int imgIndex, int imoffset, unsigned lati, int setoffset)

  Starts the annotation of the images on the artificial data (labels in the image name).

1.3 AnnotationsHandle::ASSIGNED Struct Reference

Shows which id from the old annotations is assigned to which id from the new annotations based on what minimal distance.

Public Member Functions

- **ASSIGNED** (const ASSIGNED &assig)
- ASSIGNED & **operator=** (const ASSIGNED &assig)

Data Fields

- short int **id_**
- short int **to_**
- float **dist_**

1.4 Auxiliary Class Reference

Static Public Member Functions

- static cv::Mat ipl2mat (IplImage *ipl_image)

  Converts a pointer to an IplImage to an OpenCV Mat.

- static IplImage * mat2ipl (const cv::Mat &image)

  Converts an OpenCV Mat to a pointer to an IplImage.

- static void normalizeMat (cv::Mat &matrix)

  Convert the values from a cv::Mat of floats to be between 0 and 1.

- static void range1Mat (cv::Mat &matrix)

  Changes the values of the matrix to be between [-1,1].

- static void mat2TxtFile (cv::Mat &matrix, char *fileName, bool append=false)

  Write a 2D-matrix to a text file (first row is the dimension of the matrix).

- static void txtFile2Mat (cv::Mat &matrix, char *fileName)

  Reads a 2D-matrix from a text file (first row is the dimension of the matrix).

- static void mat2BinFile (cv::Mat &matrix, char *fileName, bool append=false)

  Write a 2D-matrix to a binary file (first the dimension of the matrix).

- static void binFile2mat (cv::Mat &matrix, char *fileName)

  Reads a 2D-matrix from a binary file (first the dimension of the matrix).

- static std::string int2string (int i)

  Convert int to string.

- static void angle0to360 (float &angle)

  *Changes a given angle in RADIANS to be positive and between [0,2*M_PI).*

- static void angle180to180 (float &angle)

  Changes a given angle in RADIANS to be positive and between [-M_PI,M_PI).

- static void perpendicularLine (const cv::Point2f &A, const cv::Point2f &B, const cv::Point2f &C, float &m, float &b)

  Get perpendicular to a line given by 2 points A,B in point C.

- static bool sameSubplane (const cv::Point2f &test, const cv::Point2f &point, float m, float b)

  Checks to see if a point is on the same side of a line like another given point.

- static void showZoomedImage (const cv::Mat &image, const std::string &title)

  Just displaying an image a bit larger to visualize it better.

- static void mean0Variance1 (cv::Mat &mat)

  A function that transforms the data such that it has zero mean and unit variance: img = (img-mean(img(:)))/std(img(:)).

- static bool isSmallerPointX (const cv::Point2f &p1, const cv::Point2f &p2)

  Used to sort a vector of points -- compares points on the X coordinate.

- static bool isLargerKey (const cv::KeyPoint &k1, const cv::KeyPoint &k2)

  Compares 2 keypoints based on their response.

- static bool isLongerContours (const std::vector< cv::Point > &c1, const std::vector< cv::Point > &c2)

  Compares 2 the lengths of 2 openCV contours (vectors of vectors of cv::Point).

- static void savePCA (const std::tr1::shared_ptr< cv::PCA > pcaPtr, const std::string &file)

  Store the PCA model locally so you can load it next time when you need it.

- static std::tr1::shared_ptr< cv::PCA > loadPCA (const std::string &file)

  Load the PCA model locally so you can load it next time when you need it.

- static void getRidOfPCA (cv::PCA *pca)

  Deallocates a PCA pointed by a pointer.

- static void mean0Variance1 (cv::Mat &mat, cv::Mat &mean, cv::Mat &var)

  Mean and stddev for matrices.

1.5 Cholesky Class Reference

The `Cholesky` decomposition is used to solve Ax = b;if A is symmetric and positive definite => we can decompose A = LL* and instead of solving Ax = b, solve Ly = b for y,and the solve L*x = y for x.

Public Member Functions

- **Cholesky** (const Cholesky &c)
- Cholesky & **operator=** (const Cholesky &c)
- void init ()

  (Re)Initializes the class variables so the same instance of the class can be used for multiple decompositions.

- bool checkDecomposition ()

  Checks to see if the decomposition was already done (returns true if it is done).

- int decomposeCov (const cv::Mat &a)

  Decomposes the (covariance) matrix A into A = LL.*

- void solve (const cv::Mat &b, cv::Mat &x)

  Solves the general linear system: Ax = b and returns x.

- void solveL (const cv::Mat &b, cv::Mat &y)

  Solve the simplified equation Ly = b,and return y (where A=LL).*

- void solveLTranspose (const cv::Mat &b, cv::Mat &y)

Solve the simplified equation L'y = b,and return y (where A=LL∗).

- void inverse (cv::Mat &ainv)

  Returns the inverse of the covariance: A^{-1}.

- _float logDet ()

  Returns the log of the determiner of the (covariance) matrix,A.

- unsigned **n** ()
- cv::Mat **covar** ()

1.6 ClassifyImages Class Reference

Class used for classifying the training data.

Public Types

- enum CLASSIFIER { GAUSSIAN_PROCESS, NEURAL_NETWORK, K_NEAREST_NEIGHBORS, DIST2PCA }

  All available uses of this class.

- enum USES { EVALUATE, BUILD_DICTIONARY, TEST, BUILD_DATA }

  All available uses of this class.

Public Member Functions

- ClassifyImages (int argc, char ∗∗argv, ClassifyImages::USES use=ClassifyImages::EVALUATE, ClassifyImages::CLASSIFIER classi=ClassifyImages::GAUSSIAN_PROCESS)

  Constructor & destructor of the class.

- ClassifyImages::USES **what** ()
- void buildDictionary (int colorSp=-1, bool toUseGT=true)

  Build dictionary for vector quantization.

- void train (AnnotationsHandle::POSE what, bool fromFolder, bool justLoad=true)

  Trains on the training data using the indicated classifier.

- void trainGP (AnnotationsHandle::POSE what, int i)

  Creates the training data (according to the options),the labels and trains the a `GaussianProcess` on the data.

- void trainNN (int i, bool together=false)

  Creates the training data (according to the options),the labels and trains the a `Neural` Network on the data.

- cv::Point2f predictGP (cv::Mat &testRow, int i)

  Creates the test data and applies `GaussianProcess` prediction on the test data.

- cv::Point2f predictNN (cv::Mat &testRow, AnnotationsHandle::POSE what, int i, bool together=false)

  Creates the test data and applies `Neural` Network prediction on the test data.

- void init (float theNoise, float theLengthSin, float theLengthCos, const std::deque< FeatureExtractor::FEATURE > &theFeature, GaussianProcess::kernelFunction theKFunction=&GaussianProcess::sqexp, bool toUseGT=false)

  Initialize the options for the Gaussian Process regression.

- bool isClassiInit (int i)

  Check if the classifier was initialized.

- void evaluate (const std::deque< std::deque< cv::Point2f > > &prediAngles, float &error, float &normError, float &meanDiff)

  Evaluate one prediction versus its target.

- void crossValidation (unsigned k, unsigned fold, bool onTrain=false)

  Do k-fold cross-validation by splitting the training folder into training-set and validation-set.

- float runCrossValidation (unsigned k, AnnotationsHandle::POSE what, int colorSp=-1, bool onTrain=false, FeatureExtractor::FEATUREPART part=FeatureExtractor::WHOLE)

  Does the cross-validation and computes the average error over all folds.

- std::deque< std::deque< cv::Point2f > > runTest (int colorSp, AnnotationsHandle::POSE what, float &normError, FeatureExtractor::FEATUREPART part)

  Runs the final evaluation (test).

- float optimizePrediction (const GaussianProcess::prediction &predictionsSin, const GaussianProcess::prediction &predictionsCos)

  Try to optimize the prediction of the angle considering the variance of sin and cos.

- void resetFeatures (const std::string &dir, const std::string &imStr, int colorSp, FeatureExtractor::FEATUREPART part=FeatureExtractor::WHOLE)

  Reset the features object when the training and testing might have different calibration,background models...

- void buildDataMatrix (int colorSp=-1, FeatureExtractor::FEATUREPART part=FeatureExtractor::WHOLE)

  Just build data matrix and store it;it can be called over multiple datasets by adding the the new data rows at the end to the stored matrix.

- void loadData (const cv::Mat &tmpData1, const cv::Mat &tmpTargets1, unsigned i, cv::Mat &outData, cv::Mat &outTargets)

  Concatenate the loaded data from the files to the currently computed data.

- void getAngleLimits (unsigned classNo, unsigned predNo, float &angleMin, float &angleMax)

  Get the minimum and maximum angle given the motion vector.

- cv::Mat reduceDimensionality (const cv::Mat &data, int i, bool train, int nEigens=0, int reshapeRows=0)

  Applies PCA on top of a data-row to reduce its dimensionality.

- void getData (std::string &trainFld, std::string &annoFld, bool fromFolder, bool test, bool justLoad=false)

  Read and load the training/testing data.

- std::deque< std::deque< cv::Point2f > > predict (AnnotationsHandle::POSE what, bool fromFolder)

  Starts the threading such that each test row is generated and predicted in real time.

- std::deque< cv::Point2f > doPredict (std::tr1::shared_ptr< PeopleDetector::DataRow > dataRow, AnnotationsHandle::POSE what, bool fromFolder)

  Predicts on the test data.

- float optimizeSin2Cos2Prediction (const GaussianProcess::prediction &predictionsSin, const GaussianProcess::prediction &predictionsCos)

  Try to optimize the prediction of the angle considering the variance of sin^2 and cos^2.

- void trainKNN (AnnotationsHandle::POSE what, int i)

  Creates the training data (according to the options),the labels and trains the a kNN on the data.

- cv::Point2f predictKNN (cv::Mat &testRow, int i)

  Creates the test data and applies `kNN` prediction on the test data.

- void trainDist2PCA (AnnotationsHandle::POSE what, int i, unsigned bins=0, unsigned dimensions=1)

  Creates the training data (according to the options),the labels and builds the eigen-orientations.

- cv::Point2f predictDist2PCA (cv::Mat &testRow, AnnotationsHandle::POSE what, int i)

  Creates the test data and applies computes the distances to the stored eigen-orientations.

- cv::Mat getPCAModel (const cv::Mat &data, int i, unsigned bins)

  Backproject each image on the 4 models, compute distances and return.

- void buildPCAModels (int colorSp, FeatureExtractor::FEATUREPART part)

  Build a class model for each one of the 4 classes.

Friends

- void parameterSetting (const std::string &errorsOnTrain, const std::string &errorsOnTest, ClassifyImages &classi, int argc, char **argv, const std::deque< FeatureExtractor::FEATURE > &feat, int colorSp, bool useGt, AnnotationsHandle::POSE what, GaussianProcess::kernelFunction kernel, unsigned folds=0)

  Run over multiple settings of the parameters to find the best ones.

- void multipleClassifier (int colorSp, AnnotationsHandle::POSE what, ClassifyImages &classi, float noise, float lengthSin, float lengthCos, GaussianProcess::kernelFunction kernel, bool useGT, FeatureExtractor::FEATUREPART part)

  Combine the output of multiple classifiers (only on testing,no multiple predictions).

1.6.1 Member Function Documentation

1.6.1.1 void resetFeatures (const std::string & *dir,* const std::string & *imStr,* int *colorSp,* FeatureExtractor::FEATUREPART *part* = **`FeatureExtractor::WHOLE`**)

Reset the features_ object when the training and testing might have different calibration,background models...

1.6.1.2 std::deque< std::deque< cv::Point2f > > predict (AnnotationsHandle::POSE *what,* bool *fromFolder*)

Predicts on the test data.

1.7 compareImg Struct Reference

Checks the image name (used to find the corresponding labels for each image).

Public Member Functions

- **compareImg** (std::string image)
- bool **operator()** (AnnotationsHandle::FULL_ANNOTATIONS anno) const
- **compareImg** (const compareImg &comp)
- compareImg & **operator=** (const compareImg &comp)

Data Fields

- std::string **imgName_**

1.8 PeopleDetector::DataRow Struct Reference

Structure to store the existing/detected locations.

Public Member Functions

- **DataRow** (const cv::Point2f &exi, unsigned int grNo, std::string name, const cv::Mat &row, const cv::Mat &targ)
- **DataRow** (const DataRow &exi)
- DataRow & **operator=** (const DataRow &exi)

Data Fields

- std::string **imgName_**
- cv::Point2f **location_**
- unsigned int **groupNo_**
- cv::Mat **testRow_**
- cv::Mat **testTarg_**

1.9 PeopleDetector::Existing Struct Reference

Structure to store the existing/detected locations.

Public Member Functions

- **Existing** (const cv::Point2f &exi=cv::Point2f(0, 0), unsigned int grNo=0)
- **Existing** (const Existing &exi)
- Existing & **operator=** (const Existing &exi)

Data Fields

- cv::Point2f **location_**
- unsigned int **groupNo_**

1.10 FeatureExtractor Class Reference

Extracts the actual features from the images and stores them in data matrix.

Data Structures

- struct keyDescr

  Structure for storing keypoints and descriptors.

- struct people

  Structure containing images of the size of the detected people.

- struct templ

  Structure to store templates so they don't get recomputed all the time.

Public Types

- enum FEATUREPART { TOP, BOTTOM, WHOLE, HEAD }

  What values can be used for the feature part to be extracted.

- enum FEATURE {
  EDGES, GABOR, HOG, IPOINTS,
  RAW_PIXELS, SIFT, SIFT_DICT, SURF,
  TEMPL_MATCHES, SKIN_BINS }

  All available feature types.

- enum ROTATE { MATRIX, TEMPLATE, KEYS }

  What needs to be rotated.

Public Member Functions

- void init (const std::deque< FeatureExtractor::FEATURE > &fType, const std::string &featFile, int colorSp, int invColorSp, FeatureExtractor::FEATUREPART part)

  Initializes the class elements.

- void reset ()

  Resets the variables to the default values.

- void initSIFT (const std::string &dictName, unsigned means=500, unsigned size=128)

  Initializes the settings for the SIFT dictionary.

- void extractFeatures (cv::Mat &image, const std::string &sourceName)

  Creates a data matrix for each image and stores it locally.

- cv::Mat extractPointsGrid (cv::Mat &image)

  Extract the interest points in a gird and returns them.

- cv::Mat extractEdges (cv::Mat &image)

  Extract edges from the whole image.

- cv::Mat extractGabor (cv::Mat &image)

  Convolves the whole image with some Gabors wavelets and then stores the results.

- cv::Mat extractSIFT (cv::Mat &image, const std::vector< cv::Point2f > &templ, const cv::Rect &roi)

  Extracts SIFT features from the image and stores them in a matrix.

- cv::Mat extractSURF (cv::Mat &image)

  Extracts all the surf descriptors from the whole image and writes them in a matrix.

- cv::Mat getTemplMatches (bool flip, const FeatureExtractor::people &person, const FeatureExtractor::templ &aTempl, const cv::Rect &roi)

  Gets the plain pixels corresponding to the upper part of the body.

- cv::Mat getHOG (bool flip, const FeatureExtractor::people &person, const FeatureExtractor::templ &aTempl, const cv::Rect &roi)

  Gets the HOG descriptors over an image.

- cv::Mat getEdges (bool flip, cv::Mat &feature, const FeatureExtractor::people &person, const cv::Rect &roi, const FeatureExtractor::templ &aTempl, float rotAngle, bool contours=false)

  Gets the edges in an image.

- cv::Mat getSURF (bool flip, cv::Mat &feature, const std::vector< cv::Point2f > &templ, const cv::Rect &roi, const cv::Mat &test, std::vector< cv::Point2f > &indices)

  SURF descriptors (Speeded Up Robust Features).

- cv::Mat getSIFT (bool flip, const cv::Mat &feature, const std::vector< cv::Point2f > &templ, const cv::Rect &roi, const cv::Mat &test, std::vector< cv::Point2f > &indices, bool oneClass=true)

  Compute the features from the SIFT descriptors by doing vector quantization.

- cv::Mat getPointsGrid (bool flip, const cv::Mat &feature, const cv::Rect &roi, const FeatureExtractor::templ &aTempl, const cv::Mat &test)

  Creates a "histogram" of interest points + number of blobs.

- cv::Mat getGabor (bool flip, cv::Mat &feature, const cv::Mat &thresholded, const cv::Rect &roi, const cv::Size &foregrSize, const FeatureExtractor::templ &aTempl, float rotAngle, int aheight)

  Convolves an image with a Gabor filter with the given parameters and returns the response image.

- cv::Mat getRawPixels (bool flip, const FeatureExtractor::people &person, const FeatureExtractor::templ &aTempl, const cv::Rect &roi, bool color=true)

  Gets the raw pixels corresponding to body of the person +/- background pixels.

- void createGabor (cv::Mat &gabor, float *params=NULL)

  Creates a gabor with the parameters given by the parameter vector.

- cv::Mat getDataRow (int imageRows, const FeatureExtractor::templ &aTempl, const cv::Rect &roi, const FeatureExtractor::people &person, const std::string &imgName, cv::Point2f &absRotCenter, cv::Point2f &rotBorders, float rotAngle, bool flip, std::vector< cv::Point2f > &keys)

  Returns the row corresponding to the indicated feature type.

- void rotate2Zero (float rotAngle, FeatureExtractor::ROTATE what, const cv::Rect roi, cv::Point2f &rotCenter, cv::Point2f &rotBorders, std::vector< cv::Point2f > &pts, cv::Mat &toRotate)

  Rotate a matrix/a template/keypoints wrt to the camera location.

- unsigned readNoMeans ()

  Return number of means.

- std::string readDictName ()

  Return name of the SIFT dictionary.

- unsigned setImageClass (unsigned aClass)

  Sets the image class and resets the dictionary name.

- void getThresholdBorderes (int &minX, int &maxX, int &minY, int &maxY, const cv::Mat &thresh)

  Find the extremities of the thresholded image.

- cv::Mat cutAndResizeImage (const cv::Rect &roiCut, const cv::Mat &img)

  Cut the image around the template or bg bordered depending on which is used and resize to a common size.

- cv::Mat getSkinBins (bool flip, const FeatureExtractor::people &person, const FeatureExtractor::templ &aTempl, const cv::Rect &roi)

  Get skin/non-skin ratio of the foreground area.

- cv::Mat grabCutImage (bool flip, const FeatureExtractor::templ &aTempl, const cv::Mat &thresh, const cv::Rect &roi, const cv::Mat &feature)

  Gets the threshold/template extremities and calls cutAndResize on the input image.

Static Public Member Functions

- static bool compareDescriptors (const FeatureExtractor::keyDescr &k1, const FeatureExtractor::keyDescr &k2)

    Compares SURF 2 descriptors and returns the boolean value of their comparison.

- static bool isInTemplate (unsigned pixelX, unsigned pixelY, const std::vector< cv::Point2f > &templ)

    Checks to see if a given pixel is inside a template.

- static bool isFeatureIn (std::deque< FeatureExtractor::FEATURE > feats, FeatureExtractor::FEATURE feat)

    Find if a feature type is in the vector of features.

- static cv::Mat dist2 (const cv::Mat &mat1, const cv::Mat &mat2, cv::Mat &minDists, cv::Mat &minLabs)

    Computes the distance from the first matrix to the second and the position on which the minimum is found and the value of the minimum for each row.

1.11 AnnotationsHandle::FULL_ANNOTATIONS Struct Reference

Structure containing a vector of annotations for each image.

Public Member Functions

- **FULL_ANNOTATIONS** (const FULL_ANNOTATIONS &fanno)
- FULL_ANNOTATIONS & **operator=** (const FULL_ANNOTATIONS &fanno)

Data Fields

- std::string **imgFile_**
- std::deque< AnnotationsHandle::ANNOTATION > **annos_**

1.12 GaussianProcess Class Reference

Class implementing the Gaussian Process Regression.

Data Structures

- struct prediction

    A structure used to define predictions.

Public Types

- enum DISTRIBUTION {
  BETA, **GAUSS**, **GAUSS2D**, **GAUSSnD**,
  LOGGAUSSnD }

  All available distributions for the functions.

- typedef _float(GaussianProcess::∗ kernelFunction)(const cv::Mat &, const cv::Mat &, _float)

  Define a pointer to the kernel function.

Public Member Functions

- **GaussianProcess** (const GaussianProcess &rhs)
- GaussianProcess & **operator=** (const GaussianProcess &rhs)
- _float distribution (const cv::Mat &x, const GaussianProcess::DISTRIBUTION &distrib, const cv::Mat &mu, const cv::Mat &cov, _float a=0, _float b=0, _float s=0)

  Generates a selected distribution of the functions given the parameters (the mean: mu,the covariance: cov,the data x).

- void train (cv::Mat &X, cv::Mat &y, _float(GaussianProcess::∗fFunction)(const cv::Mat &, const cv::Mat &, _float), _float sigmasq, _float length)

  Trains the Gaussian process.

- void predict (cv::Mat &x, GaussianProcess::prediction &predi, _float length)

  Returns the prediction for the test data,x (only one test data point).

- void sampleGaussND (const cv::Mat &mu, const cv::Mat &cov, cv::Mat &smpl)

  Samples an N-dimensional Gaussian.

- _float rand_normal ()

  Returns a random number from the normal distribution.

- void sample (const cv::Mat &inputs, cv::Mat &smpl)

  Samples the process that generates the inputs.

- void sampleGPPrior (_float(GaussianProcess::∗fFunction)(const cv::Mat &, const cv::Mat &, _float), const cv::Mat &inputs, cv::Mat &smpl)

  Samples the Gaussian Process Prior.

- _float **sqexp** (const cv::Mat &x1, const cv::Mat &x2, _float l=1.0)
- _float **matern05** (const cv::Mat &x1, const cv::Mat &x2, _float l=1.0)
- _float **expCovar** (const cv::Mat &x1, const cv::Mat &x2, _float l=1.0)
- _float **matern15** (const cv::Mat &x1, const cv::Mat &x2, _float l=1.0)
- _float **matern25** (const cv::Mat &x1, const cv::Mat &x2, _float l=1.0)
- void init (GaussianProcess::kernelFunction theKFunction=&GaussianProcess::sqexp)

  Initializes or re-initializes a Gaussian Process.

- _float matchShapes (const cv::Mat &x1, const cv::Mat &x2, _float l)

Useful to compute the distance between 2 edges.

- bool empty ()

  Checks to see if the Gaussian process was trained.

1.12.1 Member Function Documentation

1.12.1.1 _float distribution (const cv::Mat & *x*, const GaussianProcess::DISTRIBUTION & *distrib*, const cv::Mat & *mu*, const cv::Mat & *cov*, _float *a = 0*, _float *b = 0*, _float *s = 0*)

Generates a selected distribution of the functions given the parameters (the mean: mu,the covariance: cov,the data_ x).

1.13 FeatureExtractor::keyDescr Struct Reference

Structure for storing keypoints and descriptors.

Public Member Functions

- **keyDescr** (const keyDescr &kdescr)
- keyDescr & **operator=** (const keyDescr &kdescr)

Data Fields

- cv::KeyPoint **keys_**
- std::deque< float > **descr_**

1.14 onScanline Struct Reference

Checks to see if a pixel's x coordinate is on a scanline.

Public Member Functions

- **onScanline** (const unsigned pixelY)
- bool **operator()** (const Helpers::scanline_t line) const
- **onScanline** (const onScanline &on)
- onScanline & **operator=** (const onScanline &on)

Data Fields

- unsigned **pixelY_**

1.15 FeatureExtractor::people Struct Reference

Structure containing images of the size of the detected people.

Public Member Functions

- **people** (const people &person)
- people & **operator=** (const people &person)

Data Fields

- cv::Point2f **absoluteLoc_**
- cv::Point2f **relativeLoc_**
- std::deque< unsigned > **borders_**
- cv::Mat **pixels_**
- cv::Mat **thresh_**

1.16 PeopleDetector Class Reference

Class used for detecting useful features in the images that can be later used for training and classifying.

Data Structures

- struct DataRow

  Structure to store the existing/detected locations.

- struct Existing

  Structure to store the existing/detected locations.

Public Types

- enum CLASSES { CLOSE, MEDIUM, FAR }

  Classes/groups (wrt the camera) in which to store the image data.

Public Member Functions

- **PeopleDetector** (int argc, char **argv, bool extract=false, bool buildBg=false, int colorSp=-1, FeatureExtractor::FEATUREPART part=FeatureExtractor::WHOLE, bool flip=true)
- virtual bool doFindPerson (unsigned imgNum, IplImage *src, const vnl_vector< float > &imgVec, vnl_vector< float > &bgVec, const float logBGProb, const vnl_vector< float > &logSumPixelBGProb)

  Overwrites the `doFindPeople` function from the `Tracker` class to make it work with the feature extraction.

- bool imageProcessingMenu ()

Simple "menu" for skipping to the next image or quitting the processing.

- void allForegroundPixels (std::deque< FeatureExtractor::people > &allPeople, const IplImage *bg, float threshold)

  Get the foreground pixels corresponding to each person.

- float getDistToTemplate (const int pixelX, const int pixelY, const std::vector< cv::Point2f > &templ)

  Gets the distance to the given template from a given pixel location.

- void extractDataRow (const IplImage *oldBg, bool flip, const std::deque< unsigned > &existing=std::deque< unsigned >(), float threshVal=50.0)

  Creates on data row in the final data matrix by getting the feature descriptors.

- void fixLabels (const std::deque< unsigned > &existing, bool flip)

  For each row added in the data matrix (each person detected for which we have extracted some features) find the corresponding label.

- void templateWindow (const cv::Size &imgSize, int &minX, int &maxX, int &minY, int &maxY, const FeatureExtractor::templ &aTempl)

  Returns the size of a window around a template centered in a given point.

- void init (const std::string &dataFolder, const std::string &theAnnotationsFile, const std::deque< FeatureExtractor::FEATURE > &feat, bool test, bool readFromFolder=true)

  Initializes the parameters of the tracker.

- bool canBeAssigned (unsigned l, std::deque< float > &minDistances, unsigned k, float distance, std::deque< int > &assignment)

  Checks to see if an annotation can be assigned to a detection.

- float fixAngle (const cv::Point2f &feetLocation, const cv::Point2f &cameraLocation, float angle, bool flip)

  Fixes the angle to be relative to the camera position with respect to the detected position.

- float unfixAngle (const cv::Point2f &headLocation, const cv::Point2f &feetLocation, float angle)

  Un-does the rotation with respect to the camera.

- void templateExtremes (const std::vector< cv::Point2f > &templ, std::deque< float > &extremes, int minX=0, int minY=0)

  Get template extremities (if needed,considering some borders -- relative to the ROI).

- void templatePart (int k, FeatureExtractor::people &person)

  If only a part needs to be used to extract the features then the threshold and the template need to be changed.

- float motionVector (const cv::Point2f &head, const cv::Point2f &center, bool flip, bool &moved)

  Computes the motion vector for the current image given the tracks so far.

- float opticalFlow (cv::Mat &currentImg, cv::Mat &nextImg, const std::vector< cv::Point2f > &keyPts, const cv::Point2f &head, const cv::Point2f &center, bool maxOrAvg, bool flip)

  Compute the dominant direction of the SIFT or SURF features.

- void keepLargestBlob (cv::Mat &thresh, const cv::Point2f &center, float tmplArea)

    Keeps only the largest blob from the thresholded image.

- void readLocations (bool flip)

    Reads the locations at which there are people in the current frame (for the case in which we do not want to use the tracker or build a bgModel).

- void start (bool readFromFolder, bool useGT)

    Starts running something (either the tracker or just mimics it).

- void add2Templates ()

    Adds a templates to the vector of templates at detected positions.

- void pixels2Templates (int maxX, int minX, int maxY, int minY, int k, const cv::Mat &thresh, float tmplHeight, cv::Mat &colorRoi)

    Assigns pixels to templates based on proximity.

- float rotationAngle (const cv::Point2f &headLocation, const cv::Point2f &feetLocation)

    Return rotation angle given the head and feet position.

- void fixLocationsTracksBorderes (const std::deque< unsigned > &existing, bool flip)

    Fixes the existing/detected locations of people and updates the tracks and creates the bordered image.

- void initInvColoprSp ()

    Initialize the inverse value of the color space used in feature extraction.

- PeopleDetector::CLASSES findImageClass (const cv::Point2f &feet, const cv::Point2f &head, bool oneClass=true)

    Find the class in which we can store the current image (the data is split in 3 classes depending on the position of the person wrt camera).

- float distanceWRTcamera (const cv::Point2f &feet)

    Get distance wrt the camera in the image.

- cv::Mat reduceDimensionality (const cv::Mat &data, int nEigens=0, int reshapeRows=0)

    Applies PCA on top of a data-row to reduce its dimensionality.

- void extractHeadArea (int i, FeatureExtractor::people &person)

    Extracts a circle around the predicted/annotated head positon.

- std::vector< cv::Mat > **data** ()
- std::vector< cv::Mat > **targets** ()
- std::deque< std::deque< float > > **dataMotionVectors** ()
- std::tr1::shared_ptr< FeatureExtractor > **extractor** ()
- void **setFlip** (bool flip)
- void drawPredictions (const cv::Point2f &pred, std::tr1::shared_ptr< PeopleDetector::DataRow > dataRow)

    Draws the target orientation and the predicted orientation on the image.

- std::tr1::shared_ptr< PeopleDetector::DataRow > popDataRow ()

Returns the last element in the data vector.

- unsigned dataInfoSize ()

  Returns the data info size.

Static Public Attributes

- static boost::mutex dataMutex_

  Used to check if the data is produced or not.

- static bool dataIsProduced_

  It is true if the data is produced.

1.17 GaussianProcess::prediction Struct Reference

A structure used to define predictions.

Public Member Functions

- **prediction** (const prediction &pred)
- prediction & **operator=** (const prediction &pred)

Data Fields

- std::deque< float > **mean_**
- std::deque< float > **variance_**

1.18 FeatureExtractor::templ Struct Reference

Structure to store templates so they don't get recomputed all the time.

Public Member Functions

- **templ** (cv::Point theCenter)
- **templ** (const templ &aTempl)
- templ & **operator=** (const templ &aTempl)

Data Fields

- cv::Point2f **center_**
- cv::Point2f **head_**
- std::deque< float > **extremes_**
- std::vector< cv::Point2f > **points_**

Index