Group Detection

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.

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 (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, std::string usedImages, int imgIndex=-1)

  Starts the annotation of 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)

  Starts the annotation of the images.

- static void drawOrientation (cv::Point2f center, unsigned int orient, annotationsHandle::POSE pose)

  Shows how the selected orientation looks on the image.

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

  Shows how the selected orientation looks on the image.

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

  Rotate matrix wrt to the camera location.

- static void writeAnnoToFile (std::deque< annotationsHandle::FULL_ANNOTATIONS > fullAnno, 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 Protected Attributes

- static IplImage * image

  The currently processed image.

- static std::deque< annotationsHandle::ANNOTATION > **annotations**
- static char choice = ' '

  Indicates if the pose was defined for the current frame.

- static boost::mutex trackbarMutex

  A mutex for controlling the access to the annotations.

- static unsigned poseSize = 5

  The number of elements in the POSE enum.

- static bool withPoses = false

  With poses or just orientation.

- static std::deque< std::string > poseNames

  The strings corresponding to the names of the poses.

1.2.1 Member Function Documentation

1.2.1.1 int runAnn (int *argc,* char ** *argv,* unsigned *step,* std::string *usedImages,* int *imgIndex* = *-1*) [static]

The parameters that need to be indicated are:

- step -- every "step"^th image is opened for annotation
- usedImages -- the folder where the annotated images are moved

- imgIndex -- the image index from which to start
- argv[1] -- name of directory containing the images
- argv[2] -- the file contains the calibration data of the camera
- argv[3] -- the file in which the annotation data needs to be stored

1.2.1.2 int runEvaluation (int *argc,* char ** *argv*) [static]

The parameters that need to be indicated are:

- argv[1] -- train file with the correct annotations;
- argv[2] -- test file with predicted annotations;

1.2.2 Field Documentation

1.2.2.1 image [static, protected]

An instance of the structure ANNOTATIONS storing the annotations for each image.

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.

Data Fields

- short int **id**
- short int **to**
- float **dist**

1.4 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

- 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 (cv::Mat a)

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

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

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

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

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

- void solveLTranspose (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.

Data Fields

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

1.5 classifyImages Class Reference

Class used for classifying the training data.

Public Types

- enum USES { EVALUATE, BUILD_DICTIONARY, TEST }

  All available uses of this class.

Public Member Functions

- classifyImages (int argc, char **argv, classifyImages::USES use=classifyImages::EVALUATE)

  Constructor & destructor of the class.

- void buildDictionary (int colorSp=CV_BGR2Lab, bool toUseGT=true)

  Build dictionary for vector quantization.

- void trainGP (annotationsHandle::POSE what)

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

- void predictGP (std::deque< gaussianProcess::prediction > &predictionsSin, std::deque< gaussianProcess::prediction > &predictionsCos, annotationsHandle::POSE what)

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

- void init (float theNoise, float theLength, featureExtractor::FEATURE theFeature, gaussianProcess::kernelFunction theKFunction=&gaussianProcess::sqexp, bool fromFolder=true, bool store=true, bool toUseGT=false)

  Initialize the options for the Gaussian Process regression.

- void evaluate (std::deque< gaussianProcess::prediction > predictionsSin, std::deque< gaussianProcess::prediction > predictionsCos, 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, int colorSp=CV_BGR2Lab, bool onTrain=false)

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

- void runTest (int colorSp=CV_BGR2Lab)

  Runs the final evaluation (test).

- float optimizePrediction (gaussianProcess::prediction predictionsSin, gaussianProcess::prediction predictionsCos)

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

- void resetFeatures (std::string dir, std::string imStr, int colorSp)

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

Protected Attributes

- peopleDetector * features

  An instance of `peopleDetector` class.

- cv::Mat trainData

  The training data matrix.

- cv::Mat testData

  The test data matrix.

- std::string trainFolder

  The folder containing the training images.

- std::string testFolder

  The folder containing the test images.

- std::string annotationsTrain

  The file contains the annotations for the training images.

- std::string annotationsTest

  The file contains the annotations for the test images.

- cv::Mat trainTargets

  The column matrix containing the train annotation data (targets).

- cv::Mat testTargets

  The column matrix containing the test annotation data (targets).

- gaussianProcess **gpCos**
- gaussianProcess **gpSin**
- float noise

  The noise level of the data.

- float length

  The length in the Gaussian Process.

- gaussianProcess::kernelFunction kFunction

  The kernel function in the Gaussian Process.

- featureExtractor::FEATURE feature

  Feature to be extracted.

- bool readFromFolder

  If the images are read from folder or from a file with image names.

- std::deque< std::string > imageList

  All images are stored in this list for cross-validation.

- std::deque< std::string > annoList

  All annotations for all images are stored in this list for cross-validation.

- unsigned foldSize

  The size of one fold in cross-validation.

- bool storeData

  If data is stored locally or not.

- std::string modelName

  The name of the model the be loaded/saved.

- classifyImages::USES what

  What should the class be used for.

- std::string testDir

  Directory in which to look for the test images & other files.

- std::string testImgString

  The letters in the image names for the test data.

- std::string trainDir

  Directory in which to look for the train images & other files.

- std::string trainImgString

  The letters in the image names for the train data.

- bool useGroundTruth

  Use the annotations' positions or use the tracker.

1.6 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

Data Fields

- std::string **imgName**

1.7 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 FEATURE {
  IPOINTS, EDGES, SIFT_DICT, SURF,
  SIFT, GABOR, PIXELS, HOG }

  All available feature types.

- enum ROTATE { MATRIX, TEMPLATE, KEYS }

  What needs to be rotated.

Public Member Functions

- void init (featureExtractor::FEATURE fType, std::string featFile)

  Initializes the class elements.

- void reset ()

  Resets the variables to the default values.

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

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

- void extractFeatures (cv::Mat image, std::string sourceName, int colorspaceCode)

  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, std::vector< cv::Point2f > templ=std::vector< cv::Point2f >(), cv::Rect roi=cv::Rect())

  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 getPixels (cv::Mat image, featureExtractor::templ aTempl, cv::Rect roi)

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

- cv::Mat getHOG (cv::Mat pixels, featureExtractor::templ aTempl, cv::Rect roi)

  Gets the HOG descriptors over an image.

- cv::Mat getEdges (cv::Mat feature, cv::Mat thresholded, cv::Rect roi, featureExtractor::templ aTempl, float rotAngle)

  Gets the edges in an image.

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

  SURF descriptors (Speeded Up Robust Features).

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

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

- cv::Mat getPointsGrid (cv::Mat feature, cv::Rect roi, featureExtractor::templ aTempl, cv::Mat test=cv::Mat())

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

- cv::Mat getGabor (cv::Mat feature, cv::Mat thresholded, cv::Rect roi, cv::Size foregrSize, float rotAngle)

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

- cv::Mat createGabor (float *params=NULL)

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

- cv::Mat getDataRow (cv::Mat image, featureExtractor::templ aTempl, cv::Rect roi, featureExtractor::people person, cv::Mat thresholded, cv::vector< cv::Point2f > &keys, std::string imgName, cv::Point2f absRotCenter, cv::Point2f rotBorders, float rotAngle)

  Returns the row corresponding to the indicated feature type.

- cv::Mat rotate2Zero (float rotAngle, cv::Mat toRotate, cv::Point2f &rotBorders, cv::Point2f rotCenter, featureExtractor::ROTATE what, std::vector< cv::Point2f > &pts)

  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.

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, std::vector< cv::Point2f > templ)

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

1.7.1 Member Function Documentation

1.7.1.1 void initSIFT (std::string *dictName,* unsigned *means* = ***500,*** unsigned *size* = ***128***)

Initializes the settings for the SIFT dictionary.

1.8 annotationsHandle::FULL_ANNOTATIONS Struct Reference

Structure containing a vector of annotations for each image.

Data Fields

- std::string **imgFile**
- std::deque< annotationsHandle::ANNOTATION > **annos**

1.9 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)(cv::Mat, cv::Mat, float)

  Define a pointer to the kernel function.

Public Member Functions

- float distribution (cv::Mat x, gaussianProcess::DISTRIBUTION distrib, cv::Mat mu=cv::Mat(), cv::Mat cov=cv::Mat(), 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)(cv::Mat, 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 (cv::Mat mu, cv::Mat cov, cv::Mat &smpl)

  Samples an N-dimensional Gaussian.

- float rand_normal ()

  Returns a random number from the normal distribution.

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

  Samples the process that generates the inputs.

- void sampleGPPrior (float(gaussianProcess::*fFunction)(cv::Mat, cv::Mat, float), cv::Mat inputs, cv::Mat &smpl)

  Samples the Gaussian Process Prior.

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

  Initializes or re-initializes a Gaussian Process.

Protected Attributes

- cholesky chlsky

  An instance of the class `cholesky`.

- cv::Mat alpha

  A variable to chace the values of alpha from the algorithm.

- cv::Mat data

  Data matrix used for training.

- unsigned N

  Number of training data points (data.rows).

- kernelFunction kFunction

  Pointer to the kernel function to be used.

- bool **_norm_fast**
- float **_norm_next**
- float **_norm_max**
- int **rand_x**
- int **rand_y**

1.10 featureExtractor::keyDescr Struct Reference

Structure for storing keypoints and descriptors.

Data Fields

- cv::KeyPoint **keys**
- std::deque< float > **descr**

1.11 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 scanline_t line) const

Data Fields

- unsigned **pixelY**

1.12 featureExtractor::people Struct Reference

Structure containing images of the size of the detected people.

Data Fields

- cv::Point2f **absoluteLoc**
- cv::Point2f **relativeLoc**
- std::deque< unsigned > **borders**
- cv::Mat_< cv::Vec3b > **pixels**

1.13 peopleDetector Class Reference

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

Public Types

- enum FEATUREPART { TOP, BOTTOM, WHOLE }

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

Public Member Functions

- **peopleDetector** (int argc, char **argv, bool extract=false, bool buildBg=false)
- 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, std::deque< unsigned > existing, IplImage *bg, float threshold)

  Get the foreground pixels corresponding to each person.

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

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

- void extractDataRow (std::deque< unsigned > &existing, IplImage *bg)

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

- std::deque< unsigned > fixLabels (std::deque< unsigned > existing)

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

- void templateWindow (cv::Size imgSize, int &minX, int &maxX, int &minY, int &maxY, featureExtractor::templ aTempl, int tplBorder=100)

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

- void init (std::string dataFolder, std::string theAnnotationsFile, featureExtractor::FEATURE feat, 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 (cv::Point2f feetLocation, cv::Point2f cameraLocation, float angle)

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

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

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

- void templatePart (cv::Mat &thresholded, int k, float offsetX, float offsetY)

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

- float motionVector (cv::Point2f head, cv::Point2f center)

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

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

  Compute the dominant direction of the SIFT or SURF features.

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

  Keeps only the largest blob from the thresholded image.

- std::deque< unsigned > readLocations ()

  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 (std::deque< unsigned > existing)

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

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

  Assigns pixels to templates based on proximity.

- cv::Point2f headLocation (cv::Point2f center)

  Gets the location of the head given the feet location.

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

  Return rotation angle given the head and feet position.

Data Fields

- bool print

  To print some feature values or not.

- bool plot

  If it is true it displays the tracks of the people in the images.

- cv::Mat data

  The training data obtained from the feature descriptors.

- cv::Mat **targets**
- std::deque< annotationsHandle::FULL_ANNOTATIONS > **targetAnno**
- unsigned lastIndex

  The previous size of the data matrix before adding new detections.

- int colorspaceCode

  The colorspace code to be used before extracting the features.

- peopleDetector::FEATUREPART featurePart

  Indicates if the part from the image to be used (feet, head, or both).

- unsigned tracking

  If the data is sequential motion information can be used.

- cv::Mat entireNext

  The the previous image.

- bool onlyExtract

  If only the features need to be extracted or the data.

- bool useGroundTruth

  Use ground truth to detect the people instead.

- featureExtractor * extractor

  An instance of the class featureExtractor.

- std::string datasetPath

  The path to the dataset to be used.

- std::string imageString

  The string that appears in the name of the images.

- std::vector< featureExtractor::templ > **templates**

1.13.1 Field Documentation

1.13.1.1 data

The targets/labels of the data.

1.14 gaussianProcess::prediction Struct Reference

A structure used to define predictions.

Data Fields

- std::deque< float > **mean**
- std::deque< float > **variance**

1.15 featureExtractor::templ Struct Reference

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

Public Member Functions

- **templ** (cv::Point theCenter)

Data Fields

- cv::Point2f **center**
- cv::Point2f **head**
- std::deque< float > **extremes**
- std::vector< cv::Point2f > **points**

Index