CODE DOCUMENTATION

Submitted by

MAYANK ROY

Under the guidance of

DR. SUMANTRA DUTTA ROY

PROF. SANTANU CHAUDHURY

at

Programme for Autonomous Robotics Lab

Indian Institute of Technology – Delhi

New Delhi – 110016

Acknowledgement

Foremost I would like to thank Prof. Santanu Chaudhury for giving me this opportunity to work at PAR Lab IIT Delhi. His guidance and mentorship kept me efforts directed and focussed. Working with him has helped me grow technically and professionally. His meticulous work ethic and high standards motivated me to expand my limits. I am extremely grate full to Dr. Sumantra Dutta Roy for his support and his understanding nature. He allowed me to experiment with my ideas and try various approaches. I also thank Prof. Saha for his caring advice and for sharing his knowledge and expertise.

I would like to thank Ms. Shraddha Chaudhury, PhD scholar, IIT Delhi and Mr. Riby Abraham Boby, PhD scholar, IIT Delhi for mentoring me and taking me through the initial stages at PAR lab. Their guidance helped me pick up new skills and build on existing knowledge. I would also like to extend my gratitude to the assistance and guidance received from various members of PAR lab and Mechatronics lab – Mr. K. Giri, Mr. Mangal Sharma, Mr. Vinoth, Mr. Dheeraj, Mr. Sushobhit, Mr. Ashutosh, Mr. Zubair, Mr. Siddharth, Mr. Sachin, Mr. Aamir. Their helping and friendly outlook made work more fun and enjoyable.

Table of Contents

1. Introduction

In many manufacturing environments, work-pieces are supplied in plates. It is a common industrial problem to load machines with such pieces. Often human labour is employed to load an automatic machine with work-pieces. Such jobs are monotonous and do not enrich human life. While the cost of labour is increasing, human performance remains essentially constant. Furthermore, the environment in manufacturing areas is generally unhealthy. Also, when work-pieces are inserted into machines by hands limbs are often exposed to danger. This is where the concept of process automation using computer vision plays a vital role. Use of computer vision in such cases is an obvious solution. The images of a work space from the camera are processed and transformed into computer commands to control the robot for performing the task at hand.

Motivation

Industries generally rely on robots for automation to save time and money. One reason is that when humans do the same task repetitively they feel uninterested and their efficiency to perform the task efficiently decreases over time. Another reason is the cost of human labour. Robots are designed in a manner that they can perform these repetitive tasks with the same precision in lesser time. Pick and place of object is one of the essential tasks in industries. After manufacturing a product, one has to pick the manufactured object from plates/bin and organize them for shipment or storage. One solution is to use manual labour for the whole process, where labourers pick objects and organize them suitably. Another solution can be that humans arrange those objects in a tray and pass it to a robot. A robot then picks them one by one and places them in suitable fashion. Robot can be trained to go to particular locations in a tray again and again and humans can arrange the objects in the tray accordingly. But the use of a robot becomes inevitable when the workspace environment is not suitable for humans. Nuclear reactor and hot furnace are some of the few examples of such environment. One cannot survive after a specified exposure to radioactive materials. So for the handling of such materials we need the help of robot. Our motivation comes from similar situations, where no kind of human intervention is possible and the whole process needs to be automated. This will not only save time, but will also save humans from being exposed to unhealthy environment.

1.1 Setup

• Robot manipulator – KUKA KR5

• Basler Cameras.

• Micro Epsilon ScanCONTROL.(Laser Scanner)

• Suction Gripper.

• High-end computational platform – Intel Xenon 3.7 GHz. for controlling cameras and laser range finders, and interfacing to the actual robotic arm for the tasks.

1.2 Problem Statement

This project address picking and manipulating pellets in a cluttered environment using a robotic manipulator with the help of cameras. Vision is an important requirement in such a task: this is the most important sensing mechanism for us humans as well. The inputs from cameras guide the manipulator and the gripper to perform the task at hand. The motivation is to have an automatic system to perform the task. This is not an easy job, since it involves the synergistic combination of the two important but practical fields of computer vision and robotics. Computer vision is used to first identify the object of interest, and then estimate its position and orientation. After picking it, the pellet will be placed into a hollow cylindrical tube whose exact location may have not be known prior. In this case too, identification of the target location, its orientation, and then finally, using visual guidance to enable the manipulator (which has picked up the pellet with the gripper) to place the pellet into the hollow cylindrical tube. This task requires great precision, and accuracy. Vision is a natural aid in such a synergistic task. In this work, we present a solution to automate the whole process of bin picking. Following problems are addressed in this work:

- Segmentation and estimation of orientation of isolated objects placed in front of a robot using both geometric information from 3D map and intensity data form image of cluttered (multi layered) objects.

- Pickup and placement of segmented object in desired orientation while assuring collision free operation.

1.3 Methodology

These are the various **scientific and technical challenges** that were resolved in the course of this work –

Step 1 : TCP IP protocol for intercommunication b/w KUKA, Camera, Laser Scanner.

Step 2 : Multi-Threaded process initiation and handling.
Implemented for communication in background and parallel processing.

Step 3 : KUKA Position detection and command(KRL) communication from PC.

Step 4 : Camera Calibration.

Step 5 : Laser - Scanning speed

Step 6 : Camera, Laser and KUKA to World Coordinates – frame calibration.

Step 7 : Sensor Fusion

Step 8 : 3D segmentation - Image segmentation and cylinder detection.

Step 9 : Collision avoidance

Step 10 : Disturbance Monitoring

Step 11 : Pick up

Step 12 : Complete Clearing of the Bin

For a brief overview user can should go through the presentation referred (APPENDIX A.3). For details on the scientific accomplishments and results, the research submission should be consulted (APPENDIX A.1). For visual presentation please see APPENDIX A. 3.

The following section discuss the code structure and documentation of the various functions.

2. C++ Code

On PC Enter the folder path for release version of the application using command line and start – pcd_read – c.

On KRL run the OrientedPickingWithScan.src code.

2.1 Code Structure:

This work is done with the aim of extracting the pose and position of the pellets from cluttered configuration. The implementation adapts to OOPS standards. We have used the CLASS concept to maintain modularity in the code making it easy to understand, re-use and edit.

The Classes provide a structure to the code. Table given below describes the brief overview of the structure followed in objects and classes with its relation with the Task.

TASK	HEADER FILE	CLASS	OBJECT	THREAD
Connection between PC and KUKA	server.h	SERVER	KUKA	ServerThread
Grabbing Image	grabbing.cpp	n/a	n/a	n/a
Taking scan and establishing connection	getProfiles_Ethernet.cpp	n/a	n/a	n/a
Main Program Thread Management	pcd_read.cpp	n/a	n/a	Main thread
Visualization of the Point cloud	segmentation _visualizer .h	PCLViewer	SeaBoat	ViewerThread
All Computations and segmentation	pellet_finder.h	PCLfinder	Pellet	PelletfinderThread
Transformations and data structures	data_manager.h	PCLdata	Boat	PelletfinderThread

** There is one more thread, i.e. **Relay Thread,** this thread is responsible for the inter process communication (IPC). Relay Thread helps in synchronizing all the threads (server, viewer and pellet finder).

From the table given above we can deduce the main structure of the code, we re-iterate the above info from a task oriented point of view, the aim is to understand the each sub category in detail so that we have a better understanding of the code, to start with.

2.2 Tasks:

1. **Connection between PC and KUKA:** A two way connection has to be established between KUKA and PC for the synchronization. Synchronization is important because code running at KUKA is dependent on the input from PC and vice versa.
2. **Grabbing Image:** For 2D information of the system, image is grabbed using the Basler camera.
3. **Taking Scan and establishing Connection:** For the 3D part of the information, a micro epsilon laser scanner is used. Firstly a port has to be checked and assigned for the reliable connection of the laser scanner to the PC, so that while taking scan point cloud is captured precisely.
4. **Visualization of the Cloud:** The data that we obtain from the laser scanner is in the form of points .It is very difficult to comprehend these points by looking at an excel sheet. Hence visualizer gives meaning to the laser data.
5. **Transformations and data structures:** In the given framework, we have three frame of reference(Tool, camera and laser).In order to have the common frame of reference for the position and orientation estimation we need to do calibration and transformations b/w frame to get the values in one common frame. For all this information is stored in the Data_manager.h**.**
6. **All computation and Segmentation:** In order to calculate the pose and position of the pellet to be picked, several computations are needed which involves extracting information from point cloud, computing normal, applying PCA etc. Further segmentation involves other set of operations to localize the object and finally determine the pose of the object.

The different functionalities implemented in the code are accessed through main code. In the main code 4 threads are spawned. These threads communicate with each other based on the conditions given by the user and all are interlinked.

2.3 Threads:

1. **Main thread:** Main function, which is the application function for this code base. It invokes all the 4 threads viz: serverthread, relaythread, finderthread, and viewerthread. After invoking it, waits for all the threads to complete the execution.
2. **Relay Thread:** Relay thread helps in the inter process communication in between all the threads that are being spawned. This thread is very important as it helps synchronize the information between the threads using start, wait and terminate conditions for the process in the operating system, based on the check conditions necessary for the threads.
3. **Server Thread:** Server Thread is used for connecting the PC with the KUKA. This thread is created and called by the main function.
4. **Pellet Finder thread:** Finder thread is responsible for all the processing of the point cloud. It mainly deals with the processing and segmentation of the point

cloud. Takes the input from camera and the laser scanner. Desamples, scales and reject the outliers of the cloud. Then starts the segmentation of the cloud. Voxelizes the cloud and create patches. The patch is then solved to find the different kind of surfaces and its pose estimation. Various flags are then set for the picking and dropping of the pellet.

5. **Viewer Thread:** Viewer thread helps in the visualization of the 3D point cloud that has been taken using the micro epsilon laser scanner.

The detailed documentation of the PC Code is in Appendix B. Code can be found at Appendix A.2.

2.4 Camera Frame Calibration:

For the calibration camera frame , we have 3 frames Camera(C), Tool (T) and Base (B).We have B_T_T(tool position with respect to the world frame(B), using the tool position and orientation given by the robot controller.) available using values from the kuka. We need to find T_T_C which serves as extrinsic parameters of the camera .To complete the camera calibration, we optimize it using Jacobian.

Step-I (using OpenCV) - To determine corner points co-ordinates and internal parameters.

1. Grab the image of checkerboard pattern using camera.
2. Determine the cross-section points on the grid using OpenCV.
3. Store the corner points of the checker board in a text file.
4. Store the camera internal parameters in the text file.

Step-II (using MATLAB) - To determine the optimization parameter and external parameters.

1. Store the position of the camera from where you want to calibrate in a matrix.
2. Convert the values of base frame(x, y, and z)(obtained using Step-I) to tool frame.
3. Divide by z(x/z; y/z) to get the values in camera frame.
4. Multiply the values with camera internal matrix, obtained using Step-I.
5. The measured values of the cross-sectional points using the camera and the laser sensor were contested against their estimate in the tool frame.
6. To minimize the errors between the real and estimated value an optimization procedure was done.

2.5 Laser Frame Calibration:

Similarly, for laser frame calibration we have 3 frames, Laser (L), Tool (T) and Base (B). We have B_T_T available using values from the kuka. To find the transformation matrix T_T_L (Tool to laser transformation), first a rough estimate of T_T_L was obtained. By positioning the laser scanner profile onto a pointed object three points were extracted from the profile in the XZ plane of the laser sensor coordinate system. The robot end-effector was adjusted so that the points have the same z

coordinate. Using the geometrical constraints imposed an initial estimate for the transformation was obtained.

Step- I

1. As mentioned, kuka co-ordinates controller position and orientation for the three different positions of the scan line were stored in a matrix.
2. Then, a pointed object was kept on the scan line and its world co-ordinates (Tool position) and laser co-ordinates (using micro epsilon GUI) were stored in a matrix.
3. Laser co-ordinated obtained in the laser frame were converted to the base frame using B_T_T and T_T_L.
4. B_T_T is fixed for the particular position of the KUKA.
5. T_T_L has to be optimized.

3. KRL Code

The KRL code is started on the KUKA embedded OS by starting the program OrientedPickingWithScan.src code. Place in the mRoy folder. The KUKA has been programmed to act as slave to the PC. The control remains with PC as it may command KUKA to perform modular tasks in any order.

3.1 Program structure:

Pickup with Orientation using 3d Scan and Vision

CODE By Mayank Roy

ROBOT NAME KUKA Robter GmbH

$SEN_PREA[7] = startflag

PC control flag

$SEN_PREA[8] = calibflag

KUKA control flag

Flag correspondence with process as commanded by PC

startflag = 0 => PC control

startflag = 1 => Home

startflag = 3 => Camera Image Grab

startflag = 5 => Laser Scanning

startflag = 7 => Pellet Pickup

startflag = 9 => Drop Pellet

calibflag = startflag + 1 => Process completed

calibflag = 0 => kuka Idle

KUKA in idle mode

$SEN_PREA[8] = 0

3.2 Function structure:

Sample structure for - Member Functions

```
Define function name
DEF name
Define Variables in beginning of the function
INT Joint
Define frame of operation
$TOOL=TOOL_DATA[16]
$BASE=BASE_DATA[13]
Acknowledge SEN_PREA[7] == 1 by setting
$SEN_PREA[8] = 1
Define operation velocity and acceleration
FOR Joint=1 to 6
$acc_axis[Joint]=30
$vel_axis[Joint]=30
ENDFOR
For PTP motion save position values in chosen frame
PTP SuctionPos C_PTP
Signal process end
$SEN_PREA[8] = 2
End of function
```

The detailed documentation of the KRL Code is in Appendix C. Code can be found at Appendix A.2

APPENDIX A

A.1 Research Publication

[1] Mayank Roy, R. A. Boby, Shraddha Chaudhary, Santanu Chaudhury, S. Dutta Roy, S. K. Saha, “Pose Estimation of Texture-less Objects in Bin Picking using Sensor Fusion”, Submitted to IROS, 2016. – Publication

A.2 Code Base

Kuka KRL Code – Code/KRL

PC C++ Code – Code/PC

A.3 Multimedia Content

Video – Multimedia/video.mp4

Presentation – Multimedia/pictureStory.pdf

APPENDIX B

B.i. pcd_read.cpp

```
Code by Mayank Roy

Tuning and edits by Shraddha Chaudhury

Comments by Deeraj

For details on internal functions – mayank.ry@gmail.com
```

```
This is the main source file for the application and initiates all threads and
functions associated with the bin Picking application
```

```
int main (int argc, char** argv)

Main function, which is the application function for this code base. It invokes all
the 4 threads viz : serverthread, relaythread, finderthread, and viewerthread. After
invoking, it waits for all the threads to complete the execution.

The four threads are associated with the various major parallel processes required for
Bin Picking. Threads keeps on running till the boat is empty.
```

```
DWORD WINAPI relayThread(LPVOID lpParam)

Relay thread is mainly responsible for the inter process communication between all
threads.
```

```
DWORD WINAPI serverThread( LPVOID lpParam )

ServerThread is used for connecting the PC with the KUKA. This thread is created and
called by the main function.

Related Header File – server.h

Related Class – server

Related Object – kuka
```

```
DWORD WINAPI viewerThread(LPVOID lpParam)

Viewer thread is mainly responsible for visualizing the point cloud.

Related Header File – segmentation_visualizer

Related Class – PCLviewer
```

Related Object – SeaBoat

DWORD WINAPI finderThread(LPVOID lpParam)

Finder thread is responsible for all the processing of the point cloud. It mainly deals with the processing and segmentation of the point cloud.

- Takes the input from the camera and the laser scanner.
- Desamples, scales and reject the outliers of the cloud.
- Then starts the segmentation of the cloud.
- Voxelizes the cloud and create patches using region Growing.
- The patch is then solved to find the different kind of surfaces and its pose estimation using localised geometric information.
- Various flags are then set for the picking and dropping of the pellet.
- Collision is monitored for pickup by modelling the mechanical constraints of the system.
- Disturbance of objects by pick up is monitored, to update affected areas within the point cloud. This allows multiple pickups within a single scan.

Related Header File – pellet_finder.h, data_manager.h

Related Class – PCLfinder, PCLdata

Related Object – pellet, boat

Various utility functions have been defined here. These functions are mostly ones that utilise data from multiple threads and header files and are best defined in a common area which has objects from all classes within its scope.

void calibrate()

Calibrates the KUKA by positioning the robotic arm to different encoded positions.

void take_image()

Function grabs an image of the boat using camera. This function internally calls grab_image which is actually responsible for grabbing the image.

void take_scan()

Scans the boat using laser scanner. This function will internally call the laser scanner specific APIs, while simultaneously keeping track of the robot movement. After running the laser scanner profiler, it will create a point cloud for further processing. The cloud is registered in the World Frame.

3D segmentation process creates cloud -> voxel -> patches in the process of finding pellets. These need to be discarded on utilisation to prevent redundancy.

```
void delete_patch()
```

Deletes a particular patch.

```
void delete_voxel()
```

Deletes a particular voxelized patch.

```
void delete_cloud()
```

Deletes the entire point cloud.

```
void get_visual()
```

Set the startViewer flag so that the viewerthread can start visuals.

```
void stop_visual()
```

Unset the startViewer flag so that the viewerthread can stop the visuals.

```
void fuse_data(int strength3D)
```

Fuse data fuses the cloud points with the camera image. To define the ROI corresponding to the patch segmented using 3D process. A point in 3D is projected onto the image plane and pixel within a given neighbourhood is associated with it.

```
void CannyThreshold(int, void*)
```

Function performing the canny edge detection algorithm.

```
void process_image(int strength3D)
```

Process_image finds the first round of edge detection using sobel operator. The second round of edge detection is done using canny edge detector which takes sobel's output as its input.

```
void detect_circles(int strength3D)
```

Detects the presence of circle using Hough circles. This function is used for standing pellet.

```
void detect_lines(int strength3D)
```

Detects the presence of lines using Houghlines transform.

This function is used to find the base of cylinder for sleeping and slanting pellet.

```
bool intersection(Point o1, Point p1, Point o2, Point p2, Point *r)
```

Finds the intersection of two lines – the axis of the cylinder and the base of the cylinder. If intersection point is found it returns true else false.

```
double distance(Point o,Point p)
```

Calculates the Euclidean distance between two points, to find the best candidates for estimated center.

```
bool circle_center(int circleNo)
```

Back projects the best center estimated using the Hough Circles function.

```
void line_center()
```

Back projects the best center estimated using the detect line and intersection function.

```
void circle_out(int strength3D)
```

Draw a circle of different colour depending on the value of strength3D. Strength of 3D is determined by size of patch.

Smaller than expected strength is 0

Equal than expected strength is 1

Larger than expected strength is 2

```
void line_out(int strength3D)
```

Draws the line with of the base of the cylinder as detected.

```
bool collision()
```

Function checks whether the KUKA has collided with any objects. Done by checking the position, and rod orientation with respect to the bin. Adjustments are made in the path if approximation is required is small – within 15 deg.

```
void queue_pellet()
```

Add the new segmented pellet to the list of pellets to be picked up.

```
void pick_pellet()
```

pick_pellet is responsible for picking up the pellet. The position and orientation of the pellet is passed on to the KUKA in order to pick the pellet up.

```
void detect_disturbance()
```

Responsible for detecting whether the boat has been displaced while picking the pellet. A new image is taken and is compared with the old image. A difference image is computed to estimate the disturbance occurred.

```
void drop_pellet()
```

drop_pellet is responsible for dropping the pellet. The dropping position of the pellet is passed on to the KUKA. Once it reaches the dropping position it will drop the pellet. The dropping position is passed on to the KUKA by the function.

B.ii. grabbing.cpp

Code by – Basler SDK

Tuning and edits by Shraddha Chaudhury and Tanya

Comments by Deeraj

For details on internal functions – contact – Shraddha_chowdhury

```
void ProcessImage(IplImage* cimg, unsigned char* pImage, int imageSizeX, int
imageSizeY )
```

Not used in the code.

```
IplImage* GrabImage()
```

Responsible for grabbing the image using the camera. This function calls the camera specific APIs to perform the desired action. The grabbed image is returned as IpImage pointer.

```
IplImage* GrabImage(int expo)
```

Not used in the code.

```
void ProcessImage_usb( IplImage* inpimg ,IplImage* cimg, int imageSizeX, int
      imageSizeY )
```

Not used in the code.

```
IplImage* GrabImage_usb2()
```

Not used in the code.

```
IplImage* GrabImage_usb()
```

Not used in the code.

```
IplImage* remove_distortion(IplImage* input)
```

Not used in the code.

```
IplImage* remove_distortion_hole(IplImage* input)
```

Not used in the code.

B.iii. get_ethernetProfiles.cpp

Code by – Micro Epsilon SDK

Tuning and edits by Mayank Roy and Shraddha Chaudhury

Comments by Deeraj

For details on internal functions – contact – Micro Epsilon

Runs the profile scanner and relays data to PC via ethernet

```
void RunProfiler(double *x_k,double *y_k,double *z_k,double *a_k,double *b_k,double
      *c_k,double *calibflag,double (*arr)[1000][1280][8], int *ctr)
```

Runs the laser scanner profiler after setting different parameters for the laser scanner. It initially sets various parameters like, shutter time, profiler configuration and then turns ON the laser scanner. Once the scanning is done, it is transmitted over the Ethernet using the function GetProfiles_Ethernet. After scanning is done, it will go back to idle state.

```
void GetProfiles_Ethernet(double *x_k,double *y_k,double *z_k,double *a_k,double
      *b_k,double *c_k,double *calibflag,double (*arr)[1000][1280][8], int *ctr)
```

Function responsible for transmitting the scanned profile over the Ethernet.

```
void OnError(const char* szErrorTxt, int iErrorValue)
```

Displays the error text.

```
void DisplayProfile(double *pdValueX, double *pdValueZ, unsigned int
uiResolution,double *x_k,double *y_k,double *z_k,double *a_k,double *b_k,double
*c_k,double *calibflag,double (*arr)[1000][1280][8], int *ctr)
```

An internal function used by GetProfiles_Ethernet() used to relay the requisite information - Displays the scanned profile.

```
void DisplayTimestamp(unsigned char *pucTimestamp)
```

Displays the timestamp of the profile.

```
std::string Double2Str(double dValue)
```

Converts a double value to a string.

B.iv. server.h

Code by AUD, Mayank Roy

Tuning and edits by Shraddha Chaudhary and Mayank Roy

Comments by Deeraj

For details on internal functions – see socket programming for windows and parsing.

Class which is used for connecting the PC with the KUKA.This performs a two way communication between the PC and the KUKA i.e. both are able to send and receive data. The time slice between send and receive is typically 12ms.

```
class Server
```

```
Server()
```

Constructor for the class Server. This mainly initializes all the variables holding input and output string and socket address etc. which are used for communication between the PC and KUKA.

```
void Connect(bool *complete)
```

Class member function which is used to connect the PC with the KUKA. The connection is established using sockets. The overview of the function can be described as:

Initialize and create the socket.

Set the socket parameters.

Bind the socket to a particular port.

Listen for the incoming connection from KUKA.

Accept the connection from KUKA.

Parses the received command from KUKA.

Also sends command to KUKA.

The time slice between send and receive is typically 12ms.

Connection is terminated once the transaction is complete.

B.v. data_manager.h

Code by Mayank Roy – mayank.ry@gmail.com

Tuning and edits by Shraddha Chaudhury

Comments by Deeraj

For details on internal functions – visit –
http://docs.pointclouds.org/trunk/group__filters.html

http://pointclouds.org/documentation/tutorials/voxel_grid.php#voxelgrid

http://pointclouds.org/documentation/tutorials/statistical_outlier.php#statistical-outlier-removal

http://pointclouds.org/documentation/tutorials/extract_indices.php#extract-indices

http://pointclouds.org/documentation/tutorials/reading_pcd.php#reading-pcd

http://pointclouds.org/documentation/tutorials/writing_pcd.php#writing-pcd

```
class PCLdata
```

The class which holds the data structures and variables for storing the cloud that is obtained through laser scanner. It also contains many member functions which is used to manipulate these class variables and data structures. These functions are used to condition the cloud data.

```
PCLdata()
```

Intializes the mechanical dimensions and creates pointers to various point clouds.

```
void createCloud( int ctr, double (*arr)[1000][1280][8])
```

Creates the point cloud from the scanned profile. It transforms the cloud from the laser co ordinates to the basic co ordinates.

```
void scale()
```

Scales the point cloud based on the mechanical dimensions and removes points not belonging to the boat interiors.

```
void  desample()
```

Filters and desamples the point cloud.

```
void  filterOuliers()
```

Filter the outliers from the point cloud based on the set, threshold value using statistical methods relying on average point density.

```
void projectPoint(double x,double y,double z,int *u,int *v )
```

Represents a single point from world co ordinate frame to camera co ordinate frame and then to projects it to the Image plane.

```
void projectCloud(pcl::PointCloud<pcl::PointXYZ>::Ptr cloud , cv::Mat boat2D)
```

Projecting the cloud points onto the image and also establishes the backward projection correspondence.

B.vi. pellet_finder.h

Code by Mayank Roy

Tuning and edits by Shraddha Chaudhary and Mayank Roy

Comments by Deeraj

For details on internal functions – visit – mayank.ry@gmail.com

http://pointclouds.org/documentation/tutorials/normal_estimation.php#normal-estimation

http://pointclouds.org/documentation/tutorials/region_growing_segmentation.php#region-growing-segmentation

Eigen for vector algebra

```
class PCLfinder
```

Class used for the finder thread. This class mainly contains the variables and member functions for localizing and segmenting the object. This is the main class which enables the manipulation of the cloud data.

```
PCLfinder()
```

Constructor for the PCLfinder class. It is mainly responsible for initializing various flags and also allocates several pointers to various variables.

```
float  voxelize(pcl::PointCloud<pcl::PointXYZ>::Ptr cloud)
```

Finding the highest point in the point cloud. This is done by filtering the point cloud in all the 3 directions i.e. x, y and z directions. This is done to avoid processing the whole cloud in one go.

```
void findPatches()
```

The voxelized cloud acts as an input to this function. This function tries to find patches (a portion) on the voxelized cloud. The normals to the voxel points are calculated.

Based on this normals a region growing segmentation algorithm is formulated which uses Normals, Smoothness and Curvature values. Using these parameters clusters are extracted.

Create patch creates a new patch from the clusters i.e the output of findPatches. This function stores the voxel points into a cloud which is the newly created patch.

```
void create_patch(int patch_num)
```

This function essentially solves the patch. Once the patch is created solvePatch is called to solve the patch. Normal estimation and principal curvature of the patch is found. Using the principal curvature values, mean curvature value is calculated.

Based on mean curvature value patch is classified into different kind of surfaces.

```
void solvePatch()
```

Solves the cylindrical patch for sleeping and slanting pellet. Patch normals and principal curvatures are calculated. Mean axis and its normal is calculated.

Min, max and mean deviation from the mean axis is calculated in direction 1 and 2.

```
void solveCylinder()
```

PCAsolver is essentially is Principal Component Analysis. By calculating the principal component, the orientation of the pellet can be estimated. Moment of Inertia is estimated first. Using moment of inertia various other features like eccentricity eigen values and eigen vectors etc are extracted from the patch.

The first eigen vector lies along the direction of maximum varience which will the orientation of the pellet.

```
void PCAsolver()
```

B.vii. segmentation_visualizer.h

Code by Mayank Roy – mayank.ry@gmail.com

Tuning and edits by Shraddha Chaudhury

Comments by Deeraj

For details on internal functions – visit – http://docs.pointclouds.org/trunk/group__visualization.html

http://pointclouds.org/documentation/tutorials/pcl_visualizer.php#pcl-visualizer

```
class PCLViewer
```

Class which is mainly used for visualizing the cloud. This class is associated with the viewer thread.

```
PCLViewer()
```

Constructor for the PCLViewer class. The constructor mainly initializes the pointers for handling the various clouds and flags of the class.

```
int start_viewer(int argc, char** argv, pcl::PointCloud<pcl::PointXYZ>::Ptr cloud,
pcl::PointCloud<pcl::PointXYZRGB>::Ptr colored_cloud,
pcl::PointCloud<pcl::PointXYZ>::Ptr voxel, pcl::PointCloud<pcl::Normal>::Ptr
cloud_normals,pcl::PointCloud<pcl::PointXYZ>::Ptr patch,
pcl::PointCloud<pcl::Normal>::Ptr
patch_normals,pcl::PointCloud<pcl::PrincipalCurvatures>::Ptr
patch_pc,pcl::PointCloud<pcl::PointXYZ>::Ptr
condensed_patch,pcl::PointCloud<pcl::PointXYZRGBNormal>::Ptr intensityCloud)
```

The member function which is responsible for the visualization of the point cloud. This function starts executing based on the status of the startViewer flag. Based on the arguments different kinds of visualization are being called.

```
Void printUsage (const char* progName)
```

Helper function for the start_viewer function in the PCLViewer class. This function displays the different arguments based on which start_viewer function will operate.

```
boost::shared_ptr<pcl::visualization::PCLVisualizer> simpleVis
(pcl::PointCloud<pcl::PointXYZ>::Ptr cloud)
```

This function is called if the argument is '-s'. Performs the simple visualization of the point cloud.

```
boost::shared_ptr<pcl::visualization::PCLVisualizer> rgbVis
(pcl::PointCloud<pcl::PointXYZRGB>::Ptr cloud)
```

RGB Visulization of the point cloud.

```
boost::shared_ptr<pcl::visualization::PCLVisualizer> customColourVis
(pcl::PointCloud<pcl::PointXYZ>::Ptr cloud,pcl::PointCloud<pcl::PointXYZ>::Ptr patch)
```

Cloud visualization with custom colour on the point cloud.

```
boost::shared_ptr<pcl::visualization::PCLVisualizer>normalsVis( pcl::PointCloud<pcl:
             :PointXYZ>::Ptr cloud, pcl::PointCloud<pcl::Normal>::Ptr normals)
```

Patch normal visualiztion with points green colour on basic point cloud and normal.

```
boost::shared_ptr<pcl::visualization::PCLVisualizer> normalsIntegratedVis
(pcl::PointCloud<pcl::PointXYZ>::Ptr cloud)
```

Integrated normals visualiztion in the point cloud.

```
boost::shared_ptr<pcl::visualization::PCLVisualizer> pciVis
(pcl::PointCloud<pcl::PointXYZ>::Ptr cloud, pcl::PointCloud<pcl::Normal>::Ptr
      normals, pcl::PointCloud<pcl::PrincipalCurvatures>::Ptr principalCurvatures)
```

Point cloud curvature and normal visualization.

```
boost::shared_ptr<pcl::visualization::PCLVisualizer> shapesVis
      (pcl::PointCloud<pcl::PointXYZRGB>::Ptr cloud)
```

Not used in this code.

```
boost::shared_ptr<pcl::visualization::PCLVisualizer> viewportsVis
(pcl::PointCloud<pcl::PointXYZRGB>::Ptr cloud, pcl::PointCloud<pcl::Normal>::Ptr
      normals1, pcl::PointCloud<pcl::Normal>::Ptr normals2)
```

Not used in this code.

```
void keyboardEventOccurred (const pcl::visualization::KeyboardEvent &event,
                            void* viewer_void)
```

Tracks keyboard event in visualizer

```
void mouseEventOccurred (const pcl::visualization::MouseEvent &event,
                         void* viewer_void)
```

Tracks mouse event in visualizer

APPENDIX C

Member Functions

Takes robot to default position

DEF GoHome()

Define Variables in beginning of the function

INT Joint

Define frame of operation

$TOOL=TOOL_DATA[16]

$BASE=BASE_DATA[13]

Acknowledge SEN_PREA[7] == 1 by setting

$SEN_PREA[8] = 1

Define operation velocity and acceleration

FOR Joint=1 to 6

$acc_axis[Joint]=30

$vel_axis[Joint]=30

ENDFOR

For PTP motion save position values in chosen frame

SuctionPos.X = 600

SuctionPos.Y = 50

SuctionPos.Z = 200

SuctionPos.A = 0

SuctionPos.B = 0

SuctionPos.C = 0

;SuctionPos.S = 010

;SuctionPos.T = 000010

```
PTP SuctionPos C_PTP

Signal process end
$SEN_PREA[8] = 2

End of function
END



Take robot to Camera Home
DEF CameraHome()
$SEN_PREA[8] = 3
$SEN_PREA[8] = 4
END

Aligns boat to according to position taught to KUKA
DEF Calibr8()

Initiate Task
$SEN_PREA[8] = -1

Intermediate Position
$SEN_PREA[8] = -2

End Flag
$SEN_PREA[8] = -3

Wait for PC to regain control
WHILE($SEN_PREA[7]<>0)
ENDWHILE
```

```
END
```

Takes robot to Camera Home and turns on flash

```
DEF GrabImage()
CameraHome()
$SEN_PREA[8] = 3
$SEN_PREA[8] = 4
END
```

Take KUKA to start point of scan, waits for PC to start scanner, translates scanner linearly along the length of the boat, signals end of scan.

```
DEF LaserScan()
$SEN_PREA[8] = 5
PTP_REL{x 270};Increment the position in x.
$SEN_PREA[8] = 6
WHILE($SEN_PREA[7]<>0)
ENDWHILE
END
```

Pick Pellet from boat by reading position and Orientation as set by PC

```
DEF PickPellet()
$SEN_PREA[8] = 7
```

Read Position to command PTP to a dynamically estimated value as sent by PC

```
pelletPos.X = $SEN_PREA[1]
pelletPos.Y = $SEN_PREA[2]
pelletPos.Z = $SEN_PREA[3]
pelletPos.A = $SEN_PREA[4]
pelletPos.B = $SEN_PREA[5]
pelletPos.C = $SEN_PREA[6]
```

```
;Achieve Position + 200*Z
$SEN_PREA[8] = 8
WHILE($SEN_PREA[7]<>0)
ENDWHILE
END
```

```
Drop Pellet in desired position as per orientation
DEF DropPellet()
$SEN_PREA[8] = 9
$SEN_PREA[8] = 10
WHILE($SEN_PREA[7]<>0)
ENDWHILE
END
```