Rob 195 - Automated Object Detection in a Collaborative Robot Workspace

Bachelor Thesis

Degree course:	Micro and Medical Technologies
Author:	Aeschlimann Dario
Tutors:	Prof. Dr. Gabriel Gruener, Prof. Dr. Sarah Dégallier Rochat
Constituent:	AHB Biel
Experts:	Dr. Aigner Nikita
Date:	02.07.2019

Berner Fachhochschule | Haute école spécialisée bernoise | Bern University of Applied Sciences

Management Summary

This is the Bachelor Thesis "Rob 195 - Automated Object Detection in a Collaborative Robot Workspace" in the degree course Micro and Medical Technologies at the Bern University of Applied Sciences. The intention of the thesis is to improve the safety in collaborative robotic systems by detecting and avoiding collisions between the robot and objects in its workspace.

This has been realised by creating an occupancy grid of the robots workspace, using two cameras to monitor the robots workspace. By simulating the planned robot movements inside the occupancy grid by casting rays between the current robot position and the goal position and checking for occupied cells in its way.

The project has been realized using the Fanuc CR-35iA cobot. As camera the Asus Xtion PRO LIVE has been chosen. The program is written in C++ using mainly the Point Cloud Library and the Octomap Library. Programs which are running on the robot were programmed using Roboguide, Fanuc's programming Software.

The function of the programm has been tested on a pick and place application where the robot has to move two objects on the table. Collisions are avoided by moving over the objects obstructing the path.

For an industrial use, a more stable interface (e.g. Ethernet) needs to be used to connect the cameras to the system. Also more than two cameras are suggested in order to avoid shadows from the robot over any potential obstacle inside the workspace and thus losing necessary data to prevent a collision. The collision avoidance system further needs to be improved to detect collision for the whole robot body and not only for the tool centre point.

Contents

1. Introduction

Collaborative robots are meant to be flexible and easy to be reprogrammed so that they can be used efficiently in the modern industrial environment. One of the most important aspects for collaborative robots is safety. The robot shall not cause any harm to people or material. Currently there are three different types [8] how the workspace of robots is monitored and protected from collisions:

- Safety monitored stop: Scanners detect humans and objects inside the workspace of the robot and force a stop of any robot motion. This means the robot is not collaborative.
- Speed and separation monitoring: Scanners detect humans and objects inside certain areas around the robot and adjust the speed of the robot when the human comes closer to the robot. This leads to a stop of any robot motion when the human is coming too close to the robot. This is also referred to as cooperative operation.
- Power and force limiting: The robot is equipped with force and torque sensor to detect any abnormal forces applied to the robot body. Detecting such a force leads to a stop any robot motion. This type is called collaborative robot.Collisions can also be detected via a sensitive skin on the robot. The skins can be pressure sensisitive or capacitive. The latter can detect flesh (they typically can not differentiate between a living human and a sausage) within a cm or two next to the skin.

Robots that rely on force sensors to detect collisions only recognize them and thus stopping any motion when the collision is already taking place. In a collaborative situation, the robot acts within a dynamic workspace and containers with liquids or heavy unstable objects may obstruct the workspace. These objects may be tipped over by the robot without triggering a halt or the halt could be triggered too late based on the force sensors, which could pose a health risk. In addition, halts caused by the force sensors cause downtime in the production process which leads

to higher production costs and longer production times, which companies like to avoid. The goal of this thesis is to develop a vision system, which detects in real-time any occupied area in the workspace, that means people or objects which are within reach of the robot, and adapts the robots trajectories to avoid any collisions, thus providing an extra layer of safety. This would allow the robot to work in a modifiable workspace together with a human and to adapt his movements according to the human ones [11].

GPU-Voxels [9] is a similar project, that already has a vision system implemented to monitor the surroundings of various robots. It is mostly used in mobile robotics but has some implementations with an collaborative robots.

Commercially available camera based systems like the Pilz SafetyEye [4] can recognize changes in the robot environment, however they do not have any capabilities to modify the robot's path.

2. Tools

In this chapter an overview of the used robot, sensors, tools and software is given. Each section contains a brief description and hardware components include a specification list. Data sheets will be attached in the appendix.

2.1. Hardware and Software

2.1.1. Fanuc CR-35iA

The Fanuc CR-35iA [1] is one of the strongest collaborative robots currently available on the market. It can lift up to 35kg and has a maximum reach of 1.8m. The robot is designed to collaborate with humans on heavy and repetitive jobs in industries like automotive, packaging, distribution and metalworking. The robot body is coated with a soft rubber skin to prevent injuries and is ISO 10218-1:2011 certified. The certification specifies requirements and guidelines for the inherent safe design, protective measures and information for use of industrial robots. The robot is equipped with a Fanuc R-30iB controller. Figure 2.1 shows the Fanuc CR-35iA besides the author for a size comparison.

Figure 2.1.: Fanuc CR-35iA robot side-by-side with a human for size comparison.

2.1.2. Asus Xtion PRO LIVE

The Asus Xtion PRO LIVE [6] (figure 2.2) camera uses infrared sensors, adaptive depth detection technology and color image sensing to capture the user's real time image and movements. It is designed for gesture and whole body detection and has a set of predefined functions to support these tasks. The camera uses USB 2.0 Interface. The distance of use is between 0.8m and 3.5m. This camera was used since it was already available multiple times in the same version, the specifications meet the requirements and it was already used in similar projects.

Figure 2.2.: Asus Xtion PRO LIVE camera

2.1.3. Roboguide

Roboguide is an offline programming software for Fanuc robots. It allows the user to create programs for the robot and simulate its workspace in 3D without the physical need and expense of a prototype workspace setup. Two different types of programs can be used with the robot. Teach pendant or TP programs are mostly used for robot motions and calling other programs. TP programs can either be written on the teach pendant or within the Roboguide software. KAREL programs are used when the TP programs reach their limits. KAREL is a language very similar to Pascal. It features strongly typed variables, constants, custom types, functions and provides all sorts of built-ins for things which can't be done with TP. Karel is a compiled language, therefor the source must be translated from a KAREL file (.kl) into p-code (.pc) which is typically done in Roboguide. Roboguide and the two programming languages were used to establish a socket communication with an external C++ program. Figure 2.3 shows the simulated workspace in Roboguide.

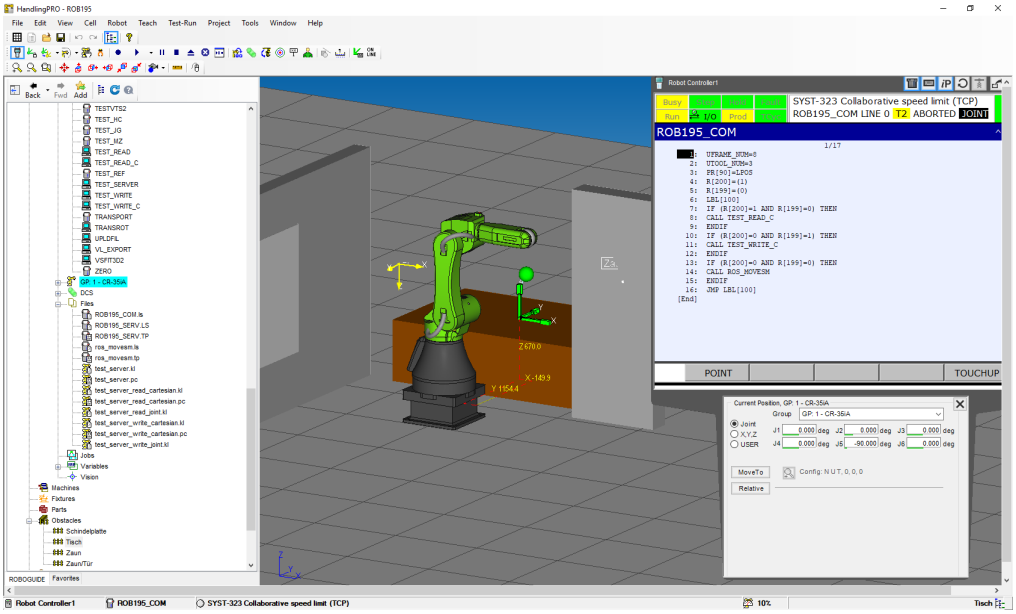


Figure 2.3.: Simulated workspace environment in Fanuc Roboguide.

2.1.4. CloudCompare

CloudCompare is a 3D point cloud processing open source software. It has been originally designed to perform comparison between two dense 3D point clouds or between a point cloud and a triangular mesh. It relies on a specific octree data structure dedicated to this task. CloudCompare has been extended to a more generic point cloud processing software, including many advanced algorithms like registration, resampling and interactive or automatic segmentation. In this project, CloudCompare is used for the camera calibration process.

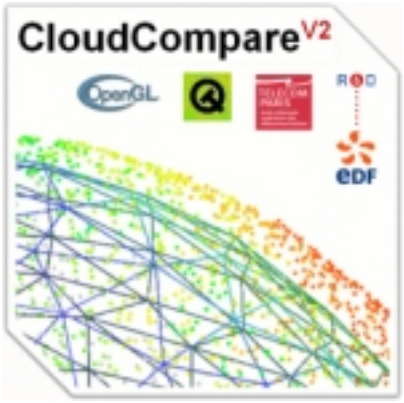


Figure 2.4.: CloudCompare logo.

2.2. Libraries and Algorithms

2.2.1. Point Cloud Library

The Point Cloud Library (PCL) is a stand-alone, large scale, open project for 2D/3D image and point cloud processing. It is released under the 3-clause BSD license and thus free for commercial and research use. The PCL framework contains numerous state-of the art algorithms including filtering, surface reconstruction, registration, model fitting and segmentation. In this project, PCL is used to process the sensor data into point clouds, including filtering and transforming.

Figure 2.5.: Point Cloud Library Logo.

2.2.2. Octomap library

The Octomap library implements a 3D occupancy grid mapping approach, providing data structures and mapping algorithms in C++ particularly suited for robotics. The map implementation is based on the octree data structure and is designed to provide a full 3D model with information about occupied, free and unknown cells. Information or new sensor readings can be added at any time and the extent of the map does not have to be known in advance. It is released under the 3-clause BSD license and thus free for commercial and research use.

3. Methods

3.1. Robot Task

A pick and place application was created to test the collision avoidance system. The pick and place task is an endless loop, where the robot uses its vacuum gripper to move two wooden shingles from one side of the table to the other side and then back again. Figure 3.1 illustrates the robot movements with arrows pointing in the direction of the robot goal position.

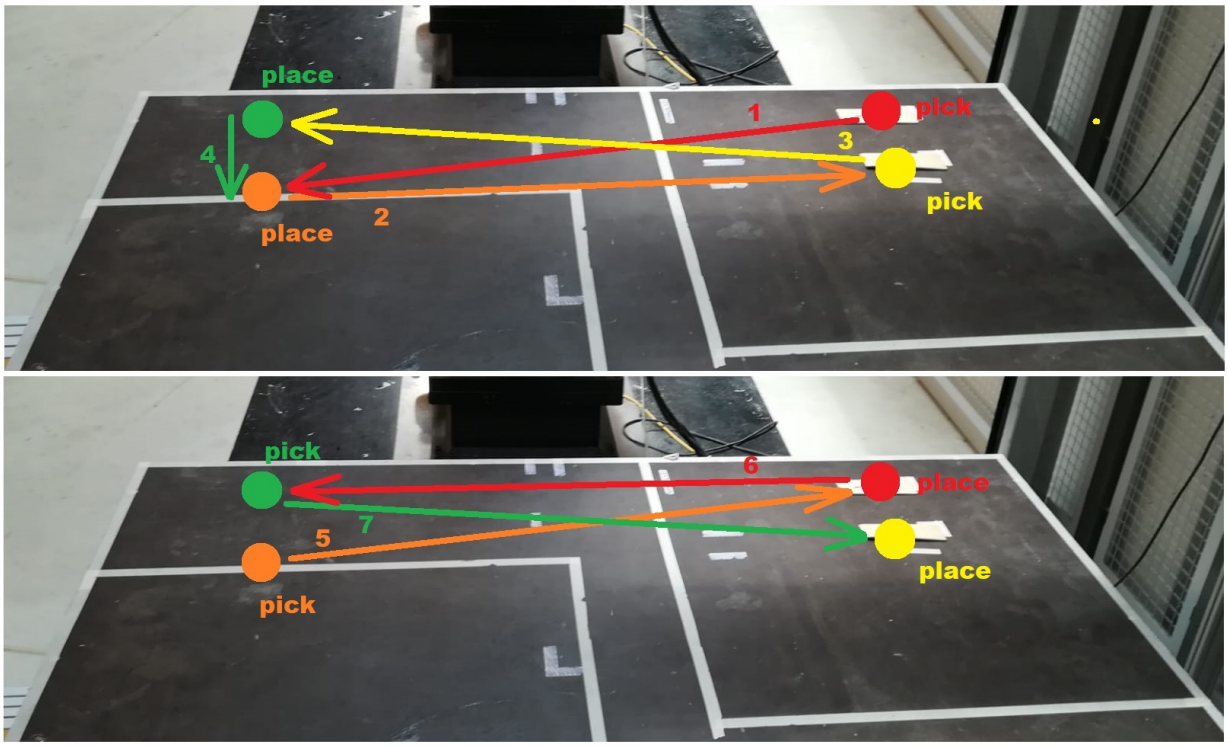


Figure 3.1.: Robot task visualised.

3.2. Robot communication

As a first task, the communication between the collision avoidance system and the robot has to be established. A sequential communication approach was chosen since it provides a stable functionality and additional control over the process. The communication on the system side is run by a single TP program calling two KAREL programs, one for reading current robot positions of the robot, and one for writing go-to positions where the robot has to move to and and additional TP program to actually move the robot. The communication on the system side is provided by two functions written in C++. The sequential communication is run on the robot and uses flags in each of the KAREL and TP programs to trigger a certain function call. Listing 3.1 shows the TP program which runs the sequential communication TP program. The user frame and the tool are defined at program start. The position register is initialized with the correct type and the flags are set in their default value. The default value is set to be ready to send the current robot position to the collision avoidance system.

```
1:  UFRAME_NUM=8 ;
2:  UTOOL_NUM=3 ;
3:  PR[90]=LPOS     ;
4:  R[200]=(1) ;
5:  R[199]=(0) ;
6:  LBL[100] ;
7:  IF (R[200]=1 AND R[199]=0) THEN ;
8:      CALL TEST_READ_C     ;
9:  ENDIF ;
10: IF (R[200]=0 AND R[199]=1) THEN ;
11:     CALL TEST_WRITE_C     ;
12: ENDIF ;
13: IF (R[200]=0 AND R[199]=0) THEN ;
14:     CALL ROS_MOVESM     ;
15: ENDIF ;
16: JMP LBL[100] ;
```

Listing 3.1: TP Program which enables the sequential communication.

3.2.1. Send and Read Robot Positions

The KAREL programs start with a header specifying different items like program name and environment settings. The KAREL programs of this project start both with almost exactly the same

header part, only differentiating in the program name. Listing 3.2 shows the file header of the read program.

```
1  PROGRAM test_read_c
2  %STACKSIZE = 4000
3  %NOLOCKGROUP
4  %NOPAUSE=ERROR+COMMAND+TPENABLE
5  %ENVIRONMENT uif
6  %ENVIRONMENT sysdef
7  %ENVIRONMENT memo
8  %ENVIRONMENT kclop
9  %ENVIRONMENT bynam
10 %ENVIRONMENT fdev
11 %ENVIRONMENT flbt
12 %ENVIRONMENT REGOPE
13 %INCLUDE klevccdf
14 %INCLUDE klevkeys
15 %INCLUDE klevkmsk
```

Listing 3.2: KAREL header of the "read robot positions" function

PROGRAM:	Specifies the program name, max. 12 characters.
%STACKSIZE = n:	Specifies the stack size in long words.
%NOPAUSE = option:	Specifies a set of conditions which will be prevented from pausing the program.
%ENVIRONMENT filename:	Used by the off-line translator to specify that a particular environment file should be loaded.
%INCLUDE filename:	Specifies files to insert into a program at translation time.

Listin 3.3 shows the second part of the KAREL program which specifies variables, which will be used during program execution.

```
VAR
file_var    : FILE
STATUS   : INTEGER
entry    : INTEGER
cur_pos: XYZWPR          -- Robot position
-- REAL Array to convert to robot position
c_real_array : ARRAY[6] OF REAL
indx    : INTEGER     --for counter

CONST
-- POS Register Number wich will be used to store the current
   robot position
MOVE_PREG = 90
-- Flags to indicate which step is next. used in ROB195_COM.ls
WRITE_FLAG = 199
READ_FLAG = 200
```

Listing 3.3: KAREL variable definitions

After the variable definitions the actual program sequence starts by setting up the server port and opening the file variable to write the current robot position as shown in listing 3.4. The port 59004 is used for the read command and port 59003 is used for the write command.

```
BEGIN
indx = 1
FORCE_SPMENU(TP_PANEL,SPI_TPUSER,1)
WRITE TPDISPLAY (CHR(128),CHR(137))
SET_FILE_ATR(file_var, ATR_IA)
-- set the server port before doing a connect
SET_VAR(entry, '*SYSTEM*',
  '$HOSTS_CFG[3].$SERVER_PORT',59004,STATUS)
WRITE TPDISPLAY('Connecting..',CR)
MSG_CONNECT('S3:',STATUS) -- connecting to system
WRITE TPDISPLAY('␣CONNECT␣STATUS␣=␣',STATUS,CR)
IF STATUS = 0 THEN -- checks connection status
-- Open S3:
        WRITE TPDISPLAY('Opening',CR)
        OPEN FILE file_var ('rw','S3:')
        STATUS = IO_STATUS(file_var)
        WRITE TPDISPLAY(STATUS,CR)
```

Listing 3.4: KAREL start connection to client.

When the connection is successful, the current robot position is written to the cur_pos variable

using the CURPOS built-in function, which returns the XYZ coordinates as well as the WPR angles. The current position needs then to be written into the real array in order to write it to the file variable and send it to the system.

```
 IF STATUS = 0 THEN
-- write an integer
        cur_pos = CURPOS(0,0)

        c_real_array[1] = cur_pos.X
        c_real_array[2] = cur_pos.Y
        c_real_array[3] = cur_pos.Z
        c_real_array[4] = cur_pos.W
        c_real_array[5] = cur_pos.P
        c_real_array[6] = cur_pos.R
```

Listing 3.5: KAREL read current robot position and prepare the data for sending.

After preparing the position data, it is sent using a for statement to loop through the REAL array and send each coordinate. Afterwards the file variable is closed and the client is disconnected. The flags are set in order to start the writing step.

```
                FOR indx = 1 TO 6 DO
                        WRITE file_var(c_real_array[indx], CR)
                ENDFOR
                CLOSE FILE file_var
        ENDIF
        WRITE TPDISPLAY('Disconnecting..',CR)
        MSG_DISCO('S3:',STATUS)
        WRITE TPDISPLAY('Done.',CR)
        SET_INT_REG(WRITE_FLAG, 1, STATUS)
        SET_INT_REG(READ_FLAG, 0, STATUS)
ENDIF
END test_read_c
```

Listing 3.6: KAREL sending position data and closing connection.

The write command is build in the same manner but reverses the steps. This means the position is written into the real array and then converted to the position variable, which is then stored in the position register of the robot controller. The code can be found in the appendix C on page A4. The program for the read and write functions is based on the socket communication example from the KAREL reference manual for the R-30iA controller [7].

The systems uses two C++ functions to either read position data from the robot or write goal positions for the robot. Listing H shows the first part of the read function, where the host address and port are set up.

```
float *connectReadCartesian (float *current_cartesian_pos){
// READ SETUP
        int sockfd;
        struct sockaddr_in serv_addr_read;
        bzero((char *) &serv_addr_read, sizeof(serv_addr_read));
        serv_addr_read.sin_family = AF_INET;
        serv_addr_read.sin_addr.s_addr = inet_addr(SERV_HOST_ADDR);
        serv_addr_read.sin_port = htons(SERV_TCP_PORT_READ);
```

Listing 3.7: C++ Set up host adress and port.

Listing 3.8 shows the second part of the read function, where the client connects to the host. When the connection is successful, the *readPos()* function is called, which reads the six parts of the robot position, each character separately.

```
// READ POS

        if((sockfd = socket(AF_INET, SOCK_STREAM,0)) < 0){
          printf("Client: Can't Open Stream Socket\n");
        }
        printf("Client: Connecting to READ server...\n");
          while(connect(sockfd,(struct sockaddr *) ...
                    &serv_addr_read, sizeof(serv_addr_read))<0){
          printf("Client: Can't Connect to the READ server\r");
        }

        printf("Client: Connected!\n");
        current_cartesian_pos = readPos(sockfd);

        return current_cartesian_pos;
}
```

Listing 3.8: C++ connect to host and read current robot position.

Writing goal positions for the robot has the same functionality as reading the positions, but again in reverse. Each character of the six coordinates separately. The code for the write function can be found in appendix F.

3.2.2. Robot movements

The robot movements are done by using the linear movement function provided by the built-in library of Fanucs TP programming language. Listing 3.9 shows the TP program, which moves the robot to the specified position in the position register #90. First it defines the tool and the frame in which the robot should move. Then the linear movement is specified by its end position, speed and if the point is a fixed point or a via point. FINE states that the point is a fixed point which has to be exactly reached. The flags R[200] and R[199] are used to trigger the next step of the sequence of the communication.

```
1 1:  UTOOL_NUM=3 ;
2 2:  UFRAME_NUM=8 ;
3 3:L PR[90] 250mm/sec FINE ;
4 5:  R[200]=(1) ;
5 6:  R[199]=(0) ;
```

Listing 3.9: TP Move the robot the the specified position.

3.2.3. Gripping commands

Usually the gripping commands would be executed using fieldbus communication. However, currently no working fieldbus communication is available on the robot.

In order to have a very simple communication to send gripping commands, two simple LUA scripts (listing 3.10) were written, each connecting to the robot controller over telnet writing a specified value to a digital out of the robot, that activates the gripper.

```
1  --set up telnet conn
2  local ip = '147.87.144.251';
3  local port = 23; -- TELNET
4  local sock = require('socket');
5  local serv, err = sock.connect(ip, port);
6  local action = 0 -- actual gripping command
7  --actuation
8  serv:send('LOGIN\r');
9  sock.sleep(0.15);
10 serv:send('PASSWORD\r');
11 sock.sleep(0.15);
12 serv:send('set␣port␣rdo[7]='.. action.. '\r');
13 sock.sleep(0.25);
14 serv:close();
```

Listing 3.10: LUA Script for telnet connection.

The two LUA scripts only differ in the value of the variable "local action" on listing 3.10 on line six. A "0" is used to disable the vacuum of the gripper, and a "1" is used to enable the gripper vacuum. In order to improve code readability, two C++ functions were implemented named gripperOn (listing 3.11)/ gripperOff (listing 3.12) located in the com_func.cpp file.

```
1 void gripperOn (void){
2         system("lua5.3␣../Rob195/1.txt");
3 }
```

Listing 3.11: gripperOn function

```
1 void gripperOff (void){
2         system("lua5.3␣../Rob195/0.txt");
3 }
```

Listing 3.12: gripperOff function

3.3. Data gathering, workspace monitoring

3.3.1. Camera positioning

The original camera design did not allow to mount the camera with screws in a fixed position as seen in figure 2.2 in section 2.1.2. In order to be able to do so, a fixture piece was designed using

the CAD Software NX [3]. The fixture allows the camera to be fixed in any desired angle. Since the fixture doesn't have any requirements regarding applied forces or stability except for holding the camera, the piece was 3D printed.

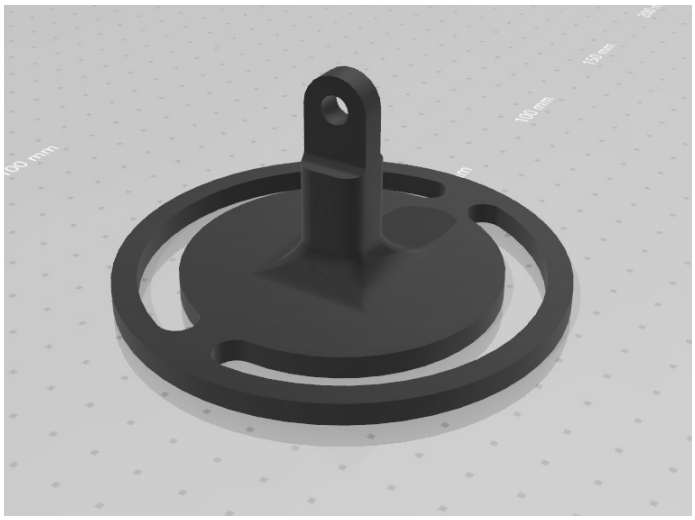


Figure 3.2.: camera fixture design.

To mount the cameras, item profiles [2] were used to build a simple structure to position the cameras.

It was not possible to monitor the whole workspace with only two cameras, so only one side of the robot workspace is monitored. The cameras were placed diagonally over the table which is placed inside the workspace as shown in Figure 3.3.

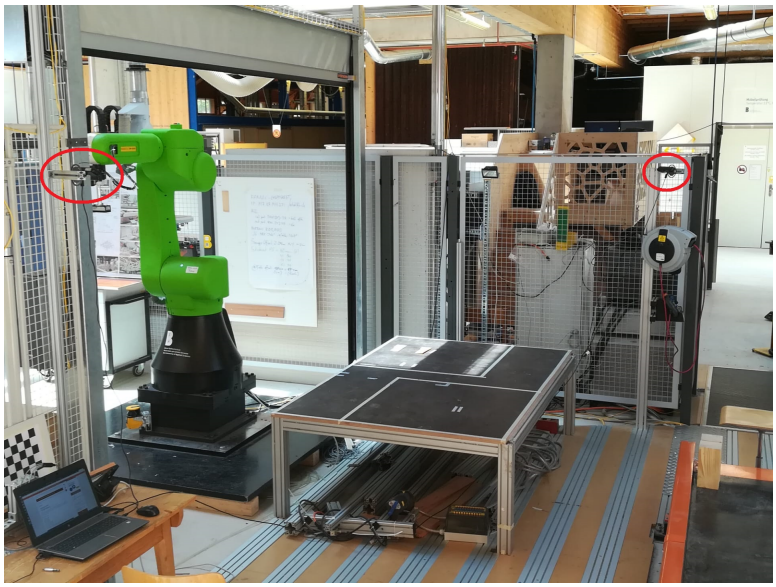


Figure 3.3.: Final camera positions, cameras are marked red.

The diagonal positioning was chosen in order to have any side of objects on the table visible in the camera view. In a first approach the cameras were set up right beside the table, but since the projected infrared pattern of each camera interferes with each other, the cameras were moved further away from each other to reduce the intensity[10] of the infrared structure and thus to reduce the interference.

The interference of the cameras resulted in loss of data, but only on horizontal surfaces. The sides of the objects were still recognised, since each side of the object is only visible by a certain camera. Figure 3.4 shows the infrared pattern and how it behaves with obstacles in it.

Figure 3.4.: Infrared structure capture. On the left side the interference, which results ins data loss, is marked in red. Green marked is the area which is inside the object shadow of one camera.

3.3.2. Camera calibration

The calibration of the cameras is done using CloudCompare and its possibility to align multiple point clouds to each other. For this a reference point cloud of the table (Figure 3.5) is generated when executing the camCalibration program. The program creates a point cloud with the surface of the table in red color. The two marked squares on the table are marked blue in the point cloud. For axis recognition a small green surface is created below the table (negative Z-direction). The table reference cloud is created at the position in the world frame of the robot.

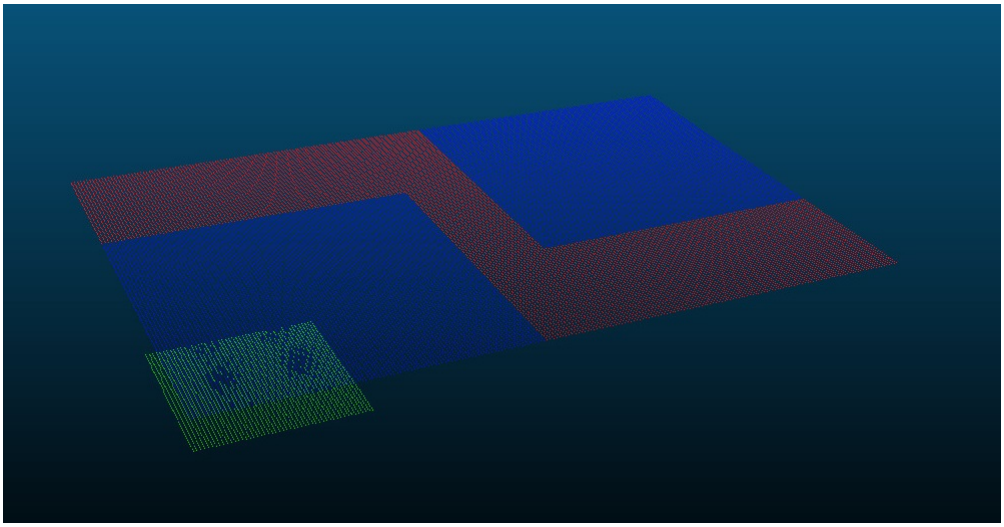


Figure 3.5.: Reference point cloud of the table surface

In addition a point cloud capture of the robot workspace from each camera is generated when the program is executed. The calibration is then done by loading the three point clouds into Cloud Compare and using the "Align two point clouds by picking (at least 4) equivalent points" command, which is located at in the top bar of Cloud Compare as marked with a red circle in figure 3.6. After clicking on the command button, a window appears in which the reference cloud can be set. Be sure to select the table reference cloud as reference.

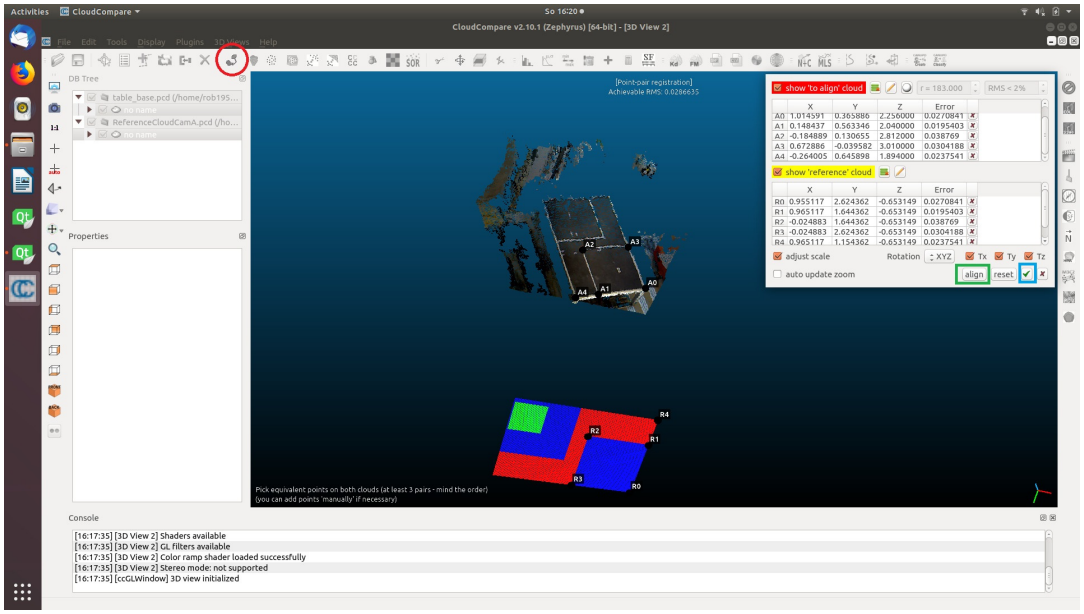


Figure 3.6.: Table reference point cloud and a single camera generated poit cloud. Red Mark shows the "align" command. After picking the points the green marked "align" button can be clicked to show the result of the command. By clicking on the blue marked "approve" button, the changes are applied.

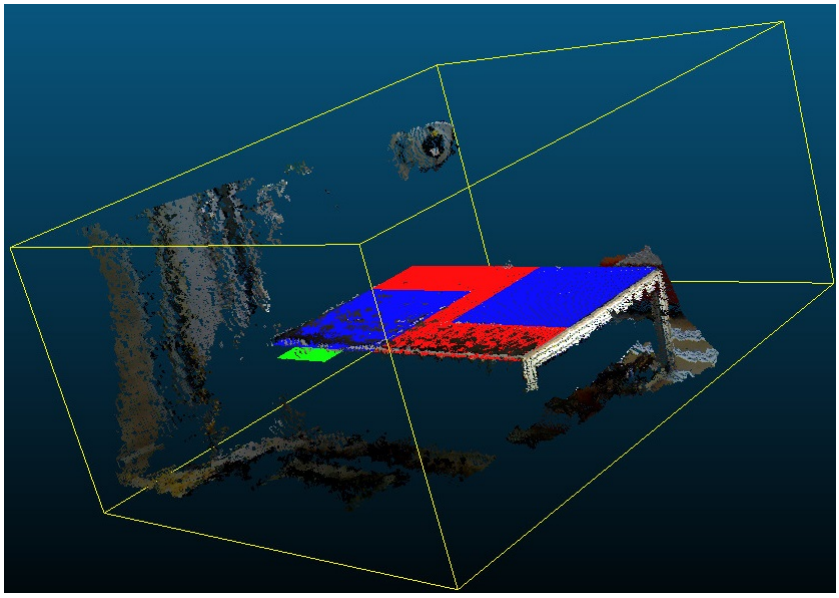


Figure 3.7.: Single camera generated point cloud aligned with the table reference cloud.

Figure 3.7 shows a single point cloud aligned with the table reference. After aligning both camera generated point clouds, the transformation matrix can be found on the properties window, after selecting the specific point cloud, on the left side of Cloud Compare. This matrix can then be copied and pasted into the "cloudA.txt" respectively the "cloudB.txt" file, located in the Software/Rob195 folder. The matrices are then read into the collision avoidance software at the start of the main function (listing 3.13).

```
// read cloud transformation data from file
ifstream fin ("./../Rob195/cloudA.txt");

if (fin.is_open()){
        for (int row = 0; row < nrows; row++){
                for (int col = 0; col < ncols; col++){
                        float item = 0.0;
                        fin >> item;
                        transformCamA(row, col) = item;
                }
        }
        fin.close();
}
ifstream fin2 ("./../Rob195/cloudB.txt");
if (fi2n.is_open()){
        for (int row = 0; row < nrows; row++){
                for (int col = 0; col < ncols; col++){
                        float item = 0.0;
                        fin2 >> item;
                        transformCamB(row, col) = item;
                }
        }
        fin2.close();
}
```

Listing 3.13: part of the main function of the collision avoidance system which reads the transformation matrices for camera A and B.

3.3.3. Data acquisition

For the data acquisition the cameras were run using the OpenNI Grabber Framework in PCL. The simple openNI viewer class is available from the tutorials of the PCL. The example file was used and adapted to support a second camera. The *run()* method of the class, creates for each camera a new OpenNIGrabber interface (listing 3.14).

```
void run (){
        //**********************************
        //         start first Xtion
        Grabber* interface = new io::OpenNI2Grabber("#1");

        boost::function<void (const pcl::PointCloud
                <pcl::PointXYZRGBA>::ConstPtr&)> f =
        boost::bind (&SimpleOpenNIViewer::cloud_cb_, this, _1);

        interface->registerCallback (f);
        interface->start();
        //**********************************
        //**********************************
        //         start second Xtion
        pcl::Grabber* interface2 = new io::OpenNI2Grabber("#2");
        boost::function<void (const pcl::PointCloud
                <pcl::PointXYZRGBA>::ConstPtr&)> f2 =
        boost::bind (&SimpleOpenNIViewer::cloud_cb_2_, this, _1);

        interface2->registerCallback (f2);
        interface2->start();
        //**********************************
```

Listing 3.14: part of the run() function of the simple openNI viewer class

pcl::PointCloud<pcl::PointXYZRGBA>::ConstPtr &cloud defines a pointer to a point cloud with color values (listing 3.15). *pcl::PointXYZ* would define a point cloud without color values.

```
void cloud_cb_ (const pcl::PointCloud
        <pcl::PointXYZRGBA>::ConstPtr &cloud)
```

Listing 3.15: Function definition of the callback function, using a pointer to a point cloud with color values.

The callback methods *cloud_cb_()* and *cloud_cb_2_()* create at first a range filter to remove data which is out of bounds. This is realised with the *pcl::PassThrough* object as shown in listing 3.16. The passthrough filter can be set up to specify in which axis the data needs to be filtered and at what range the data should be filtered. Be aware that for the set up of the filter the camera frame is used and the units used are meters. *pass.setFilterLimitsNegative (true)* could be used to inverse the filter.

```
pcl::PointCloud<pcl::PointXYZRGBA>::Ptr
  CloudCamAfiltered (new pcl::PointCloud<pcl::PointXYZRGBA> ());
 // filter element
pcl::PassThrough<pcl::PointXYZRGBA> pass;
pass.setInputCloud (cloud);
// axis definition
pass.setFilterFieldName ("z");
// range definition, points from start to endpoint are included
// in the pointcloud, the rest is filtered out.
pass.setFilterLimits (0.0, 4.1);

// pass.setFilterLimitsNegative (true);

// output cloud
pass.filter (*CloudCamAfiltered);
```

Listing 3.16: Passthrough filter for the camera input, to remove data which is out of bounds.

After filtering out points out of bounds, the point clouds are downsampled using a voxelgrid filter element. This implements the octree data structure. The octree is a tree data structure in which each internal node has exactly eight children. The root node describes a cubic bounding box which encapsulates all points from the point cloud. By dividing this cube into eight smaller cubes, each representing a certain part of the root cube, the next level of the octree is generated as shown in figure 3.8. The node is always the center point of the cube it is representing. Processing a point cloud into an octree data structure already provides a form of voxelizing. The resolution can be chosen by defining how many layers the octree data structure should have or by defining the size of the cubes at the lowest level of the octree.

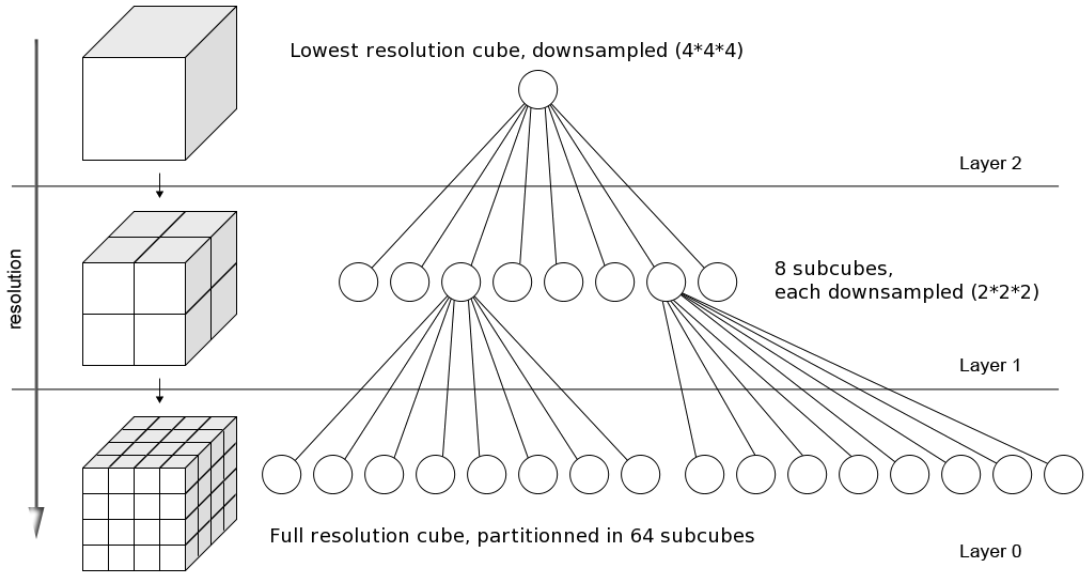


Figure 3.8.: Graphical Illustration of an octree data structure

```
1 pcl::PointCloud<pcl::PointXYZRGBA>::Ptr
2    CloudCamAVoxelGrid (new pcl::PointCloud<pcl::PointXYZRGBA> ());
3 // filter element
4 pcl::VoxelGrid<pcl::PointXYZRGBA> voxelFilter;
5 voxelFilter.setInputCloud(CloudCamAfiltered);
6 // cube size definition in meters
7 voxelFilter.setLeafSize (0.01f, 0.01f, 0.01f);
8 // output cloud
9 voxelFilter.filter (*CloudCamAVoxelGrid);
```

Listing 3.17: Voxelgrid filter to downsample the point cloud and reduce the total amount of points.

After filtering, the point clouds are transformed to their exact position by using the previously generated transform matrices. This is done by using the *pcl::transformPointCloud (*CloudCamAVoxelGrid, *transformedCloudCamA, transformCamA);* command. As parameters the input cloud, the output cloud and the transformation matrix have to be given to the function.

After transformation the point clouds are saved to the disk, when the program is executed with the debug flag (-d).

3.4. Data processing and mapping

After the point clouds are pre-filtered and transformed into the necessary frame, a single point cloud is generated by concatenating (listing 3.18) the data from camera A and B. This is done inside the *run()* function of the simpleOpenNIviewer class.

```
1 *concatenatedCloud = *transformedCloudCamA;
2 *concatenatedCloud += *transformedCloudCamB;
```

Listing 3.18: merging the point clouds from camera A and B into a single point cloud by concatenating the data.

The merged point cloud is then filtered again into the actual desired size of the cubes for the occupancy grid. The reason the merged point cloud is filtered again is simply due to avoid having data points close to each other by merging the point clouds from camera A and B. Since both cameras capture the same picture from a different angle, it is possible to have multiple points on the same spot. After applying the filter, an empty occupancy grid is created and afterwards filled with the data points from the merged point cloud. For this project a leaf size of 0.04m has been chosen since bigger leaf sizes appeared too big and inaccurate while lower leaf sizes resulted in longer calculating time. Figure 3.9 shows an occupancy grid created by the system.

```
// creating map object and defining its leaf size
octomap::ColorOcTree tree( 0.04 );
for (float x = -2.5; x <= 2.5; x += 0.08f) {
        for (float y = 0; y <= 4; y += 0.08f) {
                for (float z = -1.5; z <= 2; z += 0.08f) {
                        octomap::point3d endpoint(x, y, z);
                        // integrate 'free' measurement
                        tree.updateNode(endpoint, false);
                }
        }
}

//filling map with point cloud data
for (auto p:cloud_filtered->points){
        tree.updateNode(octomap::point3d(p.x, p.y, p.z), true);
}
// adding color information
for (auto p:cloud_filtered->points){
        tree.integrateNodeColor( p.x, p.y, p.z, p.r, p.g, p.b );
}
```

Listing 3.19: Creating an empty occupancy grid of a specified leaf size. Then filling it with th point cloud data

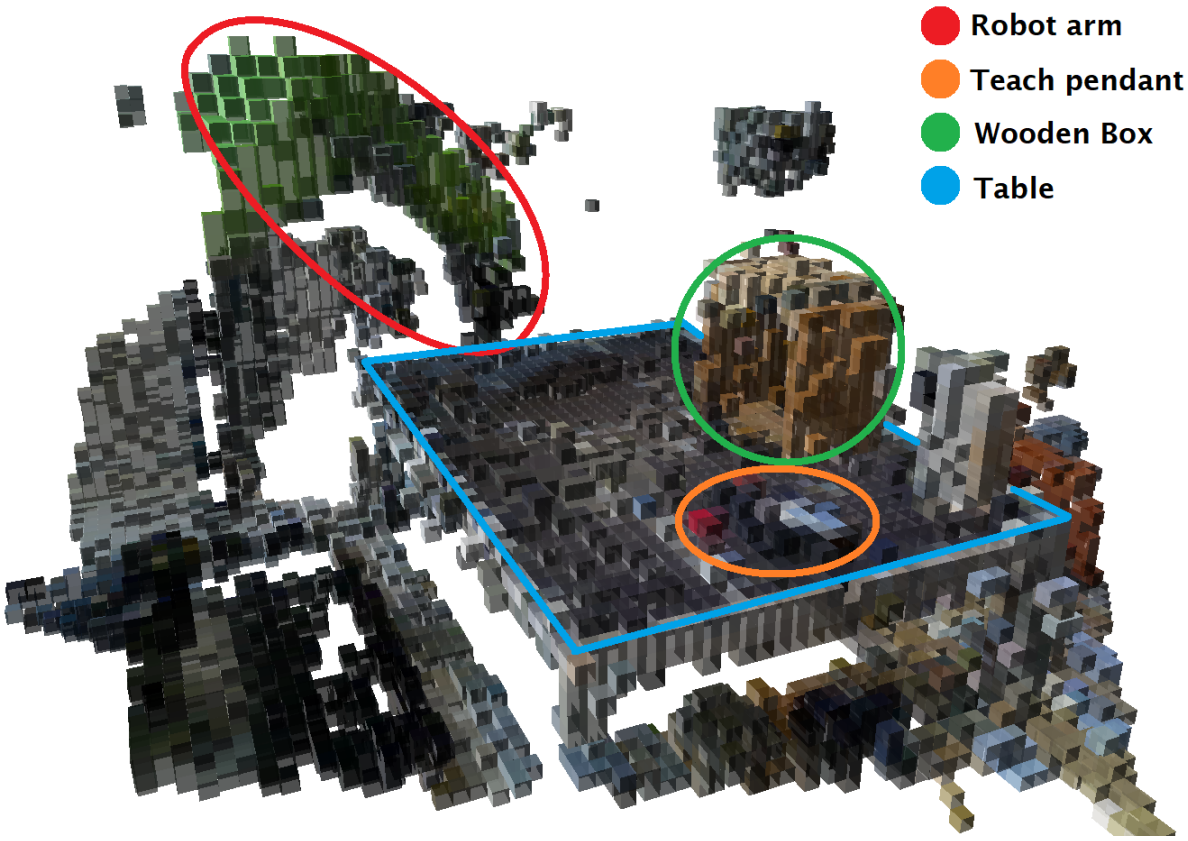


Figure 3.9.: Generated occupancy grid, already filled with the point cloud data.

3.5. Collision avoidance

The collision avoidance also takes place in the *run()* method of the viewer class. A while loop is executed as long as no safe and collision free path is found. Inside this while loop, a line is generated from the current robot position to the position where the robot should move to. The line is generated using the *octomap::ColorOctree.castRay(point3d &startingpoint, point3d &direction, point3d &endpoint, bool ignoreUknownCells, double maxRange)* function. The starting point is received from the read robot position command. It is important to know that the current position read from the robot is received in millimeters, the unit used for the raycasting in the occupancy grid is meters. Be sure to have the correct units given to the *octomap::ColorOctree.castRay()* function. The direction of the ray can be calculated with simple vector geometrics (3.1).

$$\begin{pmatrix} x_{goal} \\ y_{goal} \\ z_{goal} \end{pmatrix} - \begin{pmatrix} x_{current} \\ y_{current} \\ z_{current} \end{pmatrix} = \begin{pmatrix} x_{direction} \\ y_{direction} \\ z_{direction} \end{pmatrix} \tag{3.1}$$

The endpoint returns the center coordinates of the first occupied cell that was hit by the ray. IgnoreUnknownCells defines how unknown cells are treated. For this project, unknown cells are treated as free cells and thus the value needs to be set to true. Since the point cloud data should cover all occupied cells, every other cell should be automatically set to free. The distance the castRay has to cover can again easily be calculated with vector geometrics (3.2).

$$distance = \sqrt{(x_{goal} - x_{current})^2 + (y_{goal} - y_{current})^2 + (z_{goal} - z_{current})^2} \tag{3.2}$$

The *octomap::ColorOctree.castRay()* function returns a boolean, which is set to true if any occupied cell was hit and the maximum range was not reached. Therefore a true return values represents that an obstacle has been found. If an obstacle is found in the path of the robot, the robt moves a certain distance in z-Direction, in order to try to move over the object. After moving upwards, the castRay() starts again. To avoid reaching positions the robot can't reach, only a certain amount of increments along the z-Axis are done. After five attempts to avoid the obstacle, the program is paused and waits for a user input, approving that the object has been removed from the robot workspace.

Since the robot is captured by the cameras and therefore marked as occupied cells inside the map, any robot movement upwards would generate a collision with the robot model. To avoid this straight upwards movements do not trigger a raycasting. The same goes for downwards movements when a piece should be picked up. The movement down to the piece would generate a collision with the piece and result in a collision. This means that for the use case created for this project only horizontal movements use the raycasting to detect collisions with objects in the robot workspace. Figure 3.10 shows a graphical overview on how the collision avoidance system operates.

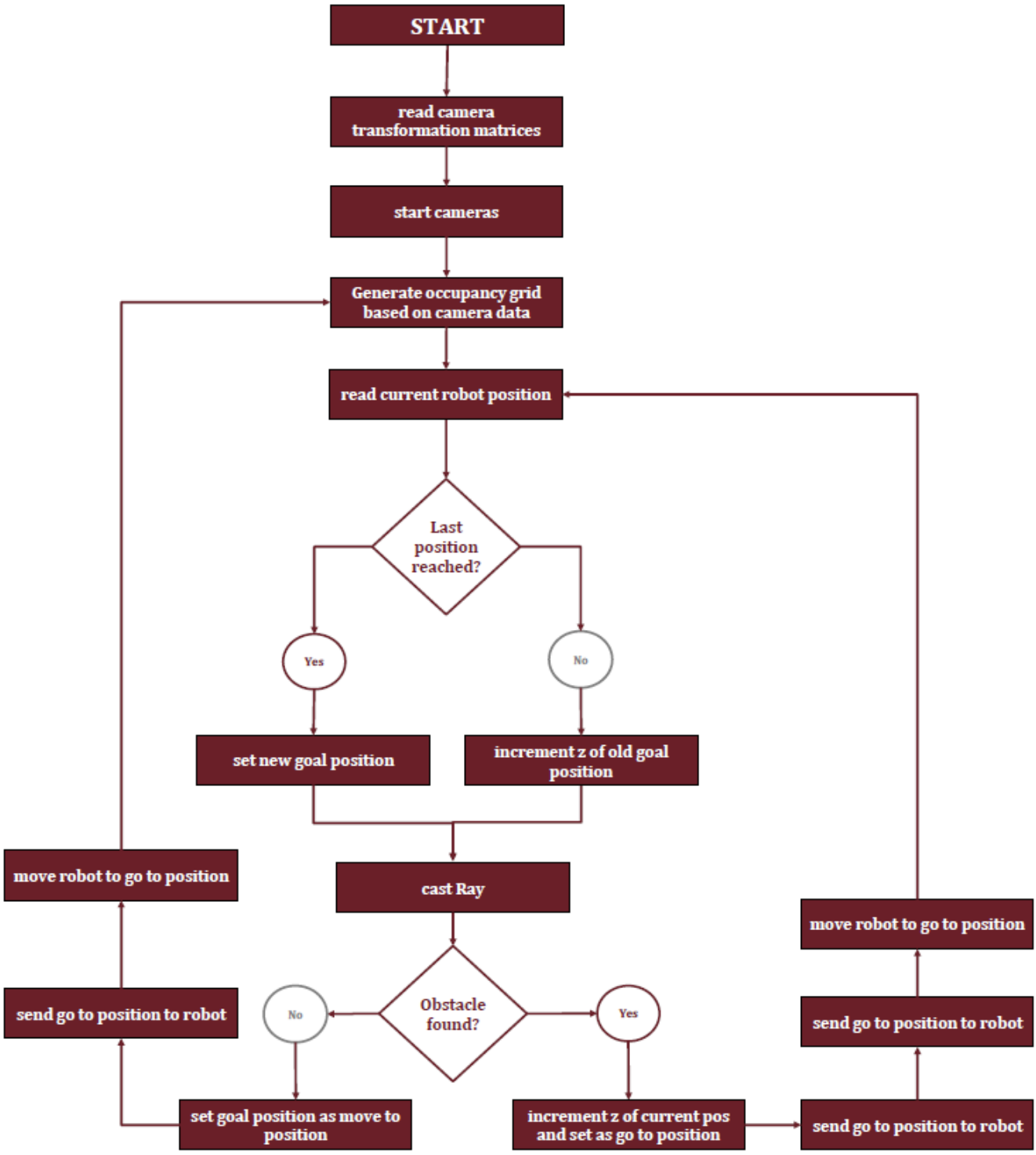


Figure 3.10.: Graphical overview of the collision avoidance system.

4. Results and outlook

Robot communication and robot movements

A first result was achieved by establishing a communication with the robot. The system is able to read the current cartesian position of the robot and write a new position into the position register of the robot controller. The communication is part of the program which runs on the robot. This program implements a sequential communication which has a strict order which needs to be followed, otherwise the system and the robot will end up not communicating with each other. As a first step to improve the collision avoidance system, a more dynamic communication should be implemented. This could be done by additionally sending communication flags from the system to the robot. These flags could then trigger the read, write or move command.

The robot movements are currently limited to linear movements, since the raycasting of movements from start to endpoint is the limiter. A way to implement joint movements could be, by dividing the joint movements in small fractions, calculate the cartesian positions of each fraction and check for collisions along these fractions before executing the joint movement.

Cameras and data acquisition

Currently, the workspace is monitored with two cameras. However, it was revealed that two cameras are not enough because if the robot is positioned in front of a camera a shadow is casted on behind objects and necessary data is lost. To avoid this data loss it is suggested to use more cameras since the infrared structure of each camera interferes with each other. it would be best if each camera can be triggered separately in order to avoid interference. Alternatively a photogrammetry based system could be used to avoid having problems with the IR pattern. A second problem encountered with the current cameras, is the unstable connection over the USB cable. This was expected since

one of the cameras needed a long (5m) cable to reach the computer where the system runs. To avoid having long USB cables, cameras with Ethernet connection are suggested.

Point cloud and map generation

The two cameras deliver a point cloud which reaches all necessary specifications in order to create an occupancy grid. The PCL would have many more functions in order to improve noise reduction and achive a more accurate mapping process. The map currently has the robot model included as occupied cells. This leads to faulty collision predictions with the robot model, especially in movements along the Z-axis. Removing the robot model from the map would solve this problem.

Collision avoidance

A simple object detection and collision avoidance was realised using the octomaps raycasting function. Therefore the system is able to detect objects in a straight path of the current position of the robot to its next position. This only regards the tool center point, collisions with any part of the robot body and the object are not detected. In order to have an industrial application it is necessary to implement a collision detection for the whole robot body. This task would need further research.

5. Conclusion

A basic approach for a collision avoidance system for collaborative robots has been programmed and partially successfully implemented.

The advantages and disadvantages of necessary tools like cameras and libraries have been evaluated and tested on a basic use case. For industrial use a better performing and more extended camera set up needs to be implemented, so the workspace can be monitored adequately.

The open source libraries used for point cloud processing and occupancy grid mapping provide the needed algorithms for this project.

The programmed use case is very basic but enables to create the most common movements of an industrial application and is therefore sufficient for this project. More important is the communication from robot to system and back. The Robots position can be sent to the system and the robots movement can be defined externally and extended with gripping commands.

Besides the movement of the robot, the data gathering of the robots workspace is key in this project. Cameras need to be mounted in the right positions to avoid creating shadows over the obstacles in the workspace and thus preventing them to be detected.

Declaration of primary authorship

I hereby confirm that I have written this thesis independently and without using other sources and resources than those specified in the bibliography. All text passages which were not written by me are marked as quotations and provided with the exact indication of its origin.

Place, Date: Biel, 02.07.2019

Last Name, First Name: Aeschlimann, Dario

Signature: ....

Bibliography

[1] Collaborative robot cr-35ia. [Online]. Available: https://www.fanuc.eu/ch/en/robots/robot-filter-page/collaborative-robots/collaborative-cr35ia

[2] Item. [Online]. Available: https://www.item24.ch/en/

[3] Nx. [Online]. Available: https://www.plm.automation.siemens.com/global/en/products/nx/

[4] Pilz, the spirit of safety. [Online]. Available: https://www.pilz.com/en-INT/eshop/00106002207042/SafetyEYE-Safe-camera-system

[5] "Process engineering." [Online]. Available: http://processengineering.co.uk/article/2020966/fanuc-extends-cobot

[6] Xtion pro live. [Online]. Available: https://www.asus.com/3D-Sensor/Xtion_PRO_LIVE/

[7] anonymous, *FANUC Robotics SYSTEM R-30iA and R-30iB Controller KAREL Reference Manual MARRC75KR07091E Rev G*.

[8] M. Bélanger-Barrette. What does collaborative robot mean ? [Online]. Available: https://blog.robotiq.com/what-does-collaborative-robot-mean

[9] A. Hermann, F. Drews, J. Bauer, S. Klemm, A. Roennau, and R. Dillmann, "Unified gpu voxel collision detection for mobile manipulation planning," 2014. [Online]. Available: http://www.gpu-voxels.org

[10] R. M. Martín, M. Lorbach, and O. Brock, "Deterioration of depth measurements due to interference of multiple rgb-d sensors."

[11] S. Rochat and G. Gruener, "Rob195 work description." [Online]. Available: https://gitlab.ti.bfh.ch/roboticsLab/Thesis/2019/R195/Documents/blob/master/spec/BFH-Rob195-Work_Specification.pdf

List of Figures

Listings

Appendices

A. Datasheet Fanuc CR-35iA

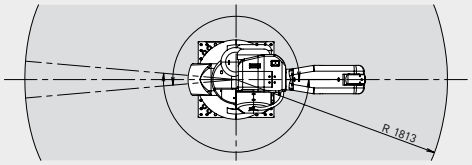
CR-35*i*A

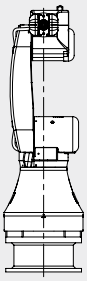
Max. load capacity at wrist: **35 kg**

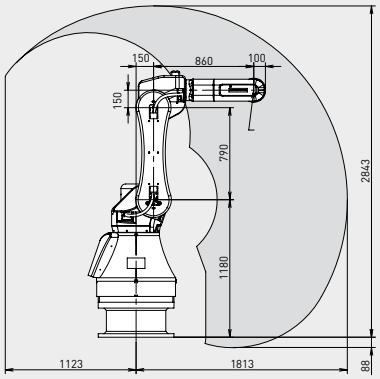
Max. reach: **1813 mm**

Controlled axes	Repeatability (mm)	Mechanical weight (kg)	Motion range (°)						Maximum speed (mm/s)*1						J4 Moment/ Inertia (Nm/kgm²)	J5 Moment/ Inertia (Nm/kgm²)	J6 Moment/ Inertia (Nm/kgm²)
			J1	J2	J3	J4	J5	J6	J1	J2	J3	J4	J5	J6			
6	± 0.03**	990	370	165	258	400	220	900	750*2						110/4	110/4	60.0/1.5

Working range







Motion range may be restricted according to the mounting angle!

Robot	**CR-35*i*A**
Robot footprint [mm]	650 x 650
Mounting position Floor	●
Mounting position Upside down	-
Mounting position Wall	-
Controller	**R-30*i*B Plus**
Open air cabinet	-
Mate cabinet	-
A-cabinet	●
B-cabinet	o
*i*Pendant Touch	●
Electrical connections	
Voltage 50/60Hz 3phase [V]	380-575
Voltage 50/60Hz 1phase [V]	-
Average power consumption [kW]	1
Integrated services	
Integrated signals on upper arm In/Out	6/4
Integrated air supply	-
Environment	
Acoustic noise level [dB]	< 70
Ambient temperature [° C]	0-45
Protection	
Body standard/optional	IP54
Wrist & J3 arm standard/optional	IP67

*1] In case of short distance motion, the speed may not reach the maximum value stated.
*2] It is necessary to set a motion speed according to risk assessment of system considering pinching with the surroundings.

● standard o on request - not available [ ] with hardware and/or software option **Based on ISO9283

MDS-00105-EN

B. Datasheet Asus Xtion PRO Live

Sensor	RGB& Depth& Microphone*2
Depth Image Size	VGA (640x480) : 30 fps QVGA (320x240): 60 fps
Resolution	SXGA (1280*1024)
Field of View	58° H, 45° V, 70° D (Horizontal, Vertical, Diagonal)
Distance of Use	Between 0.8m and 3.5m
Power Consumption	below 2.5W
Interface	USB2.0
Platform	Intel X86 & AMD
OS Support	Win 32/64 : XP , Vista, 7 Linux Ubuntu 10.10: X86,32/64 bit Android(by request)
Software	software development kits(OPEN NI SDK bundled)
Programming Language	C++/C# (Windows) C++(Linux) JAVA
Operation Environment	Indoor
Dimensions	18 x 3.5 x 5 inch

- All specifications are subject to change without notice. Please check with your supplier for exact offers. Products may not be available in all markets.
- PCB color and bundled software versions are subject to change without notice.
- Brand and product names mentioned are trademarks of their respective companies.

C. KAREL Read - code

```
1  PROGRAM test_read_c
2  %STACKSIZE = 4000
3  %NOLOCKGROUP
4  %NOPAUSE=ERROR+COMMAND+TPENABLE
5  %ENVIRONMENT uif
6  %ENVIRONMENT sysdef
7  %ENVIRONMENT memo
8  %ENVIRONMENT kclop
9  %ENVIRONMENT bynam
10 %ENVIRONMENT fdev
11 %ENVIRONMENT flbt
12 %ENVIRONMENT REGOPE
13 %INCLUDE klevccdf
14 %INCLUDE klevkeys
15 %INCLUDE klevkmsk
16 ------------------------------------------
17 VAR
18 file_var   : FILE
19 STATUS   : INTEGER
20 entry    : INTEGER
21 cur_pos: XYZWPR
22 c_real_array : ARRAY[6] OF REAL  -- REAL Array
23 indx    : INTEGER     --for counter
24
```

```
25 CONST
26
27 MOVE_PREG = 90                    -- POS Register Number wich will be us
28 WRITE_FLAG = 199
29 READ_FLAG = 200
30 ------------------------------------------
31 BEGIN
32 indx = 1
33 FORCE_SPMENU(TP_PANEL,SPI_TPUSER,1)
34 WRITE TPDISPLAY (CHR(128),CHR(137))
35 SET_FILE_ATR(file_var, ATR_IA)
36 -- set the server port before doing a connect
37 SET_VAR(entry, '*SYSTEM*','$HOSTS_CFG[3].$SERVER_PORT',59004,STATUS)
38 WRITE TPDISPLAY('Connecting..',CR)
39 MSG_CONNECT('S3:',STATUS)
40 WRITE TPDISPLAY('␣CONNECT␣STATUS␣=␣',STATUS,CR)
41 IF STATUS = 0 THEN
42 -- Open S3:
43 WRITE TPDISPLAY('Opening',CR)
44 --   FOR tmp_int1 = 1 TO 20 DO
45 OPEN FILE file_var ('rw','S3:')
46 STATUS = IO_STATUS(file_var)
47 WRITE TPDISPLAY(STATUS,CR)
48 IF STATUS = 0 THEN
49 -- write an integer
50 WRITE TPDISPLAY('Reading',CR)
51 cur_pos = CURPOS(0,0)
52 WRITE TPDISPLAY('Cartesian␣Positions:', cur_pos, CR)
53 c_real_array[1] = cur_pos.X
54 WRITE TPDISPLAY('X␣=', cur_pos.X , CR)
55 WRITE TPDISPLAY('realX␣=', c_real_array[1] , CR)
```

```
56
57 c_real_array[2] = cur_pos.Y
58 WRITE TPDISPLAY('Y␣=', cur_pos.Y , CR)
59 WRITE TPDISPLAY('realY␣=', c_real_array[2] , CR)
60
61 c_real_array[3] = cur_pos.Z
62 WRITE TPDISPLAY('Z␣=', cur_pos.Z , CR)
63 WRITE TPDISPLAY('realZ␣=', c_real_array[3] , CR)
64
65 c_real_array[4] = cur_pos.W
66 WRITE TPDISPLAY('W␣=', cur_pos.W , CR)
67
68
69 c_real_array[5] = cur_pos.P
70 WRITE TPDISPLAY('P␣=', cur_pos.P , CR)
71
72 c_real_array[6] = cur_pos.R
73 WRITE TPDISPLAY('R␣=', cur_pos.R , CR)
74
75
76 FOR indx = 1 TO 6 DO
77 --WRITE TPDISPLAY('send␣jpos␣real␣array[',indx,']:', CR)
78 --WRITE TPDISPLAY(c_real_array[indx], CR)
79 WRITE file_var(c_real_array[indx], CR)
80 ENDFOR
81
82
83 CLOSE FILE file_var
84 ENDIF
85 --ENDFOR
86 WRITE TPDISPLAY('Disconnecting..',CR)
```

```
87 MSG_DISCO('S3:',STATUS)
88 WRITE TPDISPLAY('Done.',CR)
89 SET_INT_REG(WRITE_FLAG, 1, STATUS)
90 SET_INT_REG(READ_FLAG, 0, STATUS)
91 ENDIF
92 END test_read_c
```

D. KAREL Write - code

```
1  PROGRAM test_write_c
2  %STACKSIZE = 4000
3  %NOLOCKGROUP
4  %NOPAUSE=ERROR+COMMAND+TPENABLE
5  %ENVIRONMENT uif
6  %ENVIRONMENT sysdef
7  %ENVIRONMENT memo
8  %ENVIRONMENT kclop
9  %ENVIRONMENT bynam
10 %ENVIRONMENT fdev
11 %ENVIRONMENT flbt
12 %ENVIRONMENT REGOPE
13 %INCLUDE klevccdf
14 %INCLUDE klevkeys
15 %INCLUDE klevkmsk
16 -----------------------------------------------------------------------
17 VAR
18 file_var   : FILE
19 tmp_int    : INTEGER
20 tmp_int1   : INTEGER
21 tmp_str    : STRING[128]
22 STATUS     : INTEGER
23 entry      : INTEGER
24 n_j_pos : JOINTPOS6      -- position of robot joints to go to, IMPORTAN
```

```
25 new_pos : XYZWPR
26 stat_   : INTEGER     -- status variable
27 n_real_array : ARRAY[6] OF REAL  -- REAL Array
28 indx   : INTEGER    --for counter
29 group_no: INTEGER
30 CONST
31
32 MOVE_PREG = 90                  -- POS Register Number wich will be us
33 WRITE_FLAG = 199
34 READ_FLAG = 200
35 ------------------------------------------------------------------------
36 BEGIN
37 group_no = 1
38 --Setze $UTOOL $UFRAME mit aktuell im TP eingestellten UT und UF
39 $GROUP[group_no].$UFRAME = $MNUFRAME[group_no, $MNUFRAMENUM[group_no]]
40 $GROUP[group_no].$UTOOL = $MNUTOOL[group_no, $MNUTOOLNUM[group_no]]
41 new_pos = CURPOS(0,0)
42 --SET_POS_REG(MOVE_PREG, new_pos, STATUS)
43 FORCE_SPMENU(TP_PANEL,SPI_TPUSER,1)
44 WRITE TPDISPLAY (CHR(128),CHR(137))
45 SET_FILE_ATR(file_var, ATR_IA)
46 -- set the server port before doing a connect
47 SET_VAR(entry, '*SYSTEM*','$HOSTS_CFG[4].$SERVER_PORT',59003,STATUS)
48 WRITE TPDISPLAY('Connecting..',CR)
49 MSG_CONNECT('S4:',STATUS)
50 WRITE TPDISPLAY('␣CONNECT␣STATUS␣=␣',STATUS,CR)
51 IF STATUS = 0 THEN
52 -- Open S4:
53 WRITE TPDISPLAY('Opening',CR)
54 --   FOR tmp_int1 = 1 TO 20 DO
55 OPEN FILE file_var ('rw','S4:')
```

```
56 STATUS = IO_STATUS(file_var)
57 WRITE TPDISPLAY(STATUS,CR)
58 IF STATUS = 0 THEN
59 -- write an integer
60 WRITE TPDISPLAY('Reading',CR)
61
62 -- Read 10 bytes
63
64 FOR indx = 1 TO 6 DO
65 BYTES_AHEAD(file_var, entry, STATUS)
66 READ file_var (tmp_str::126)
67 WRITE TPDISPLAY('receive:',tmp_str,CR )
68 CNV_STR_REAL(tmp_str, n_real_array[indx])
69 --              n_real_array[indx] = tmp_str
70 --              READ file_var (n_real_array[indx]::126,CR)
71 WRITE TPDISPLAY('2.receive:',n_real_array[indx],CR )
72 ENDFOR
73 new_pos.X = n_real_array[1]
74 new_pos.Y = n_real_array[2]
75 new_pos.Z = n_real_array[3]
76 new_pos.W = n_real_array[4]
77 new_pos.P = n_real_array[5]
78 new_pos.R = n_real_array[6]
79 WRITE TPDISPLAY('New␣Positions:',new_pos, CR)
80 SET_POS_REG(MOVE_PREG, new_pos, STATUS)
81
82 CLOSE FILE file_var
83 ENDIF
84
85 WRITE TPDISPLAY('Disconnecting..',CR)
86 MSG_DISCO('S4:',STATUS)
```

```
87 WRITE TPDISPLAY('Done.',CR)
88 SET_INT_REG(WRITE_FLAG, 0, STATUS)
89 SET_INT_REG(READ_FLAG, 0, STATUS)
90 ENDIF
91 END test_write_c
```

E. C++ com_func.h code

```
1  #ifndef COM_FUNC_H
2  #define COM_FUNC_H
3
4  #define MAXLINE 512
5
6  #define SERV_TCP_PORT_READ 59004
7  #define SERV_TCP_PORT_WRITE 59003
8
9  //#define SERV_HOST_ADDR "147.87.149.40" /* RoboGuide PC Robot */
10 #define SERV_HOST_ADDR "147.87.144.251" /* Robot */
11
12
13
14
15 /**
16  * @brief connectReadJoint
17  * @param current_joint_pos
18  * @return
19  */
20 float* connectReadJoint (float *current_joint_pos);
21
22 /**
23  * @brief connectReadCartesian
24  * @param current_cartesian_pos
```

```
25  * @return
26  */
27 float *connectReadCartesian (float *current_cartesian_pos);
28
29 /**
30  * @brief connectWriteJoint
31  * @param goto_joint_pos
32  * @param current_joint_pos
33  */
34 void connectWriteJoint (float *goto_joint_pos, float *current_joint_po
35
36 /**
37  * @brief connectWriteCartesian
38  * @param goto_cartesian_pos
39  * @param current_cartesian_pos
40  */
41 void connectWriteCartesian (float *goto_cartesian_pos, float *current_
42
43 /**
44  * @brief readJointPos
45  * @param sockfd
46  * @return
47  */
48 float* readPos(int sockfd);
49
50 /**
51  * @brief writeJointPos
52  * @param sockfd
53  * @param goto_pos
54  */
55 void writePos (int sockfd, float *goto_pos);
```

```
56
57 /**
58  * @brief readline
59  * @param fd
60  * @param ptr
61  * @param maxlen
62  * @param input
63  * @return
64  */
65 float readline(int fd, char *ptr, int maxlen, char input[MAXLINE+1]);
66
67 /**
68  * @brief written
69  * @param fd
70  * @param ptr
71  * @param nbytes
72  * @return
73  */
74 int written(int fd, char *ptr, int nbytes);
75
76 /**
77  * @brief gripperOn
78  */
79 void gripperOn (void);
80
81 /**
82  * @brief gripperOff
83  */
84 void gripperOff (void);
85
86 #endif // COM_FUNC_H
```

F. C++ com_func.cpp - code

```
1  #include <unistd.h>
2  #include <stdio.h>
3  #include <sys/types.h>
4  #include <sys/socket.h>
5  #include <netinet/in.h>
6  #include <arpa/inet.h>
7  #include <cstring>
8  #include <iostream>
9  #include <stdlib.h>
10 #include <string>
11 #include <chrono>
12
13 #include "com_func.h"
14 using namespace std;
15
16 float *connectReadJoint (float *current_joint_pos){
17 // READ SETUP
18
19 int sockfd;
20 struct sockaddr_in serv_addr_read;
21 bzero((char *) &serv_addr_read, sizeof(serv_addr_read));
22 serv_addr_read.sin_family = AF_INET;
23 serv_addr_read.sin_addr.s_addr = inet_addr(SERV_HOST_ADDR);
24 serv_addr_read.sin_port = htons(SERV_TCP_PORT_READ);
```

```
25
26 // READ JOINT POS
27
28 if((sockfd = socket(AF_INET, SOCK_STREAM,0)) < 0){
29 printf("Client: Can't Open Stream Socket\n");
30 }
31 printf("Client: Connecting to READ server...\n");
32 if(connect(sockfd,(struct sockaddr *) &serv_addr_read, sizeof(serv_add
33 printf("Client: Can't Connect to the READ server\n");
34 }
35 else{
36 printf("Client: Connected!\n");
37 current_joint_pos = readPos(sockfd);
38 for(int k = 0; k < 6; k++){
39 printf("joint %d pos: %f \n", k+1, current_joint_pos[k]);
40 }
41 }
42 return current_joint_pos;
43 }
44
45 float *connectReadCartesian (float *current_cartesian_pos){
46 //    auto startReadConnect = chrono::steady_clock::now();
47 // READ SETUP
48 int sockfd;
49 struct sockaddr_in serv_addr_read;
50 bzero((char *) &serv_addr_read, sizeof(serv_addr_read));
51 serv_addr_read.sin_family = AF_INET;
52 serv_addr_read.sin_addr.s_addr = inet_addr(SERV_HOST_ADDR);
53 serv_addr_read.sin_port = htons(SERV_TCP_PORT_READ);
54
55 // READ JOINT POS
```

```
56
57 if((sockfd = socket(AF_INET, SOCK_STREAM,0)) < 0){
58 printf("Client: Can't Open Stream Socket\n");
59 }
60 printf("Client: Connecting to READ server...\n");
61 while(connect(sockfd,(struct sockaddr *) &serv_addr_read, sizeof(serv_
62 printf("Client: Can't Connect to the READ server\r");
63 }
64
65 printf("Client: Connected!\n");
66 current_cartesian_pos = readPos(sockfd);
67 printf("X pos: %f \n", current_cartesian_pos[0]);
68 printf("Y pos: %f \n", current_cartesian_pos[1]);
69 printf("Z pos: %f \n", current_cartesian_pos[2]);
70 printf("W pos: %f \n", current_cartesian_pos[3]);
71 printf("P pos: %f \n", current_cartesian_pos[4]);
72 printf("R pos: %f \n", current_cartesian_pos[5]);
73
74
75 //    auto endReadConnect = chrono::steady_clock::now();
76 //    cout << "Elapsed Time in milliseconds (connect read): " << chron
77 return current_cartesian_pos;
78 }
79
80 void connectWriteJoint (float *goto_joint_pos, float *current_joint_po
81 //WRITE SETUP
82 int sockfd_write;
83 struct sockaddr_in serv_addr_write;
84 bzero((char *) &serv_addr_write, sizeof(serv_addr_write));
85 serv_addr_write.sin_family = AF_INET;
86 serv_addr_write.sin_addr.s_addr = inet_addr(SERV_HOST_ADDR);
```

```
87  serv_addr_write.sin_port = htons(SERV_TCP_PORT_WRITE);
88
89  // WRITE JOINT POS
90  if((sockfd_write = socket(AF_INET, SOCK_STREAM,0)) < 0){
91  printf("Client: Can't Open Stream Socket\n");
92  }
93  printf("Client: Connecting to WRITE server...\n");
94  if(connect(sockfd_write,(struct sockaddr *) &serv_addr_write, sizeof(s
95  printf("Client: Can't Connect to the WRITE server\n");
96  }
97  else{
98  for(int s = 0; s < 6; s++){
99  goto_joint_pos[s] = current_joint_pos[s] + 10.0f;
100 printf("future joint %d pos: %f \n", s+1, goto_joint_pos[s]);
101 }
102 writePos(sockfd_write, goto_joint_pos);
103 }
104 }
105
106 void connectWriteCartesian (float *goto_cartesian_pos, float *current_
107 //    auto startWriteConnect = chrono::steady_clock::now();
108 //WRITE SETUP
109 int sockfd_write;
110 struct sockaddr_in serv_addr_write;
111 bzero((char *) &serv_addr_write, sizeof(serv_addr_write));
112 serv_addr_write.sin_family = AF_INET;
113 serv_addr_write.sin_addr.s_addr = inet_addr(SERV_HOST_ADDR);
114 serv_addr_write.sin_port = htons(SERV_TCP_PORT_WRITE);
115
116 // WRITE JOINT POS
117 if((sockfd_write = socket(AF_INET, SOCK_STREAM,0)) < 0){
```

```
118 printf("Client: Can't Open Stream Socket\n");
119 }
120 printf("Client: Connecting to WRITE server...\n");
121 while(connect(sockfd_write,(struct sockaddr *) &serv_addr_write, sizeo
122 printf("Client: Can't Connect to the WRITE server\r");
123 }
124 printf("Client: Connected!\n");
125
126 printf("future X pos: %f \n", goto_cartesian_pos[0]);
127 printf("future Y pos: %f \n", goto_cartesian_pos[1]);
128 printf("future Z pos: %f \n", goto_cartesian_pos[2]);
129 printf("future W pos: %f \n", goto_cartesian_pos[3]);
130 printf("future P pos: %f \n", goto_cartesian_pos[4]);
131 printf("future R pos: %f \n", goto_cartesian_pos[5]);
132 writePos(sockfd_write, goto_cartesian_pos);
133 //    auto endWriteConnect = chrono::steady_clock::now();
134 //    cout << "Elapsed Time in milliseconds (connect write): " << chro
135
136 }
137
138
139 float * readPos (int sockfd)
140 {
141 //    printf("Reading current joint positions...\n");
142 char input[MAXLINE+1];
143 static float input_float[6];
144 char recvline[MAXLINE + 1];
145 for(int x=0; x < 6; x++)
146 {
147 //memset (input_float, 0, 6);
148 memset (input, 0, MAXLINE+1);
```

```
149 input_float[x] = readline(sockfd, recvline, 126, input);
150 }
151 return input_float;
152 }
153
154 void writePos (int sockfd, float *goto_pos)
155 {
156 char sendline[MAXLINE];
157 for(int y=0; y < 6; y++)
158 {
159 sprintf(sendline, "%f", goto_pos[y]);
160 //        printf("sendline %u : %s\n", y, sendline);
161
162 if(written(sockfd, sendline, 126)!=126){
163 printf("writeJointPos:written␣error␣on␣sock\n");
164 }
165 //        sleep(1);
166 }
167 }
168
169 float readline(int fd, char *ptr, int maxlen, char input[MAXLINE+1])
170 {
171 int n, rc;
172 char c;
173 for(n = 0; n < maxlen; n++){
174 if((rc = read(fd, &c, 1)) == 1){
175 input[n] = c;
176 *ptr++ = c;
177 if(c=='\n'){
178 break;
179 }
```

```
180 else if(rc== 0) {
181 if(n== 0) {
182 return (0);
183 }
184 else{
185
186 break;
187 }
188 }
189 }
190 else{
191 return (-1);
192 }
193 }
194 float f = atof(input);
195 *ptr = 0;
196 return (f);
197 }
198
199 int written(int fd, char *ptr, int nbytes)
200 {
201 int nleft, nwritten;
202 nleft = nbytes;
203 //    printf("Client: nleft size: %d \n", nleft);
204 while(nleft > 0) {
205 //        printf("Client: while write started\n");
206 nwritten = write(fd, ptr, nleft); /*was nleft*/
207 //        printf("Client: nwritten size: %d \n", nwritten);
208 //        printf("Client: write command executed!\n");
209 if(nwritten <= 0) {
210 //            printf("Client: return nwritten\n");
```

```
211 return(nwritten);
212 }
213 //         printf("Client: nleft and ptr\n");
214 nleft -= nwritten;
215 //         printf("Client: nleft size: %d \n", nleft);
216 ptr += nwritten;
217 }
218 //    printf("Client: return nbytes - nleft: %d \n", nbytes-nleft);
219 return(nbytes - nleft);
220 }
221
222
223 void gripperOn (void){
224 system("lua5.3 ../Rob195/1.txt");
225 }
226
227 void gripperOff (void){
228 system("lua5.3 ../Rob195/0.txt");
229 }
```

G. C++ main.cpp - code

```
1  #include <unistd.h>
2  #include <stdio.h>
3  #include <sys/types.h>
4  #include <sys/socket.h>
5  #include <netinet/in.h>
6  #include <arpa/inet.h>
7  #include <cstring>
8  #include <iostream>
9  #include <stdlib.h>
10 #include <string>
11 #include <math.h>
12 #include <chrono>
13
14 #include <pcl/io/openni2_grabber.h>
15 #include <pcl/visualization/cloud_viewer.h>
16 #include <pcl/io/pcd_io.h>
17 #include <pcl/io/png_io.h>
18 #include <pcl/io/ply_io.h>
19 #include <pcl/point_types.h>
20 #include <pcl/console/parse.h>
21 #include <pcl/common/transforms.h>
22 #include <pcl/point_types.h>
23 #include <pcl/filters/passthrough.h>
24 #include <pcl/compression/octree_pointcloud_compression.h>
```

```
25 #include <pcl/filters/voxel_grid.h>
26
27 #include <octomap/octomap.h>
28 #include <octomap/ColorOcTree.h>
29
30 #include "com_func.h"
31
32 #define SERV_TCP_PORT_READ 59004
33 #define SERV_TCP_PORT_WRITE 59003
34
35
36 using namespace std;
37 using namespace pcl;
38
39 pcl::PointCloud<pcl::PointXYZRGBA>::Ptr transformedCloudCamB (new pcl:
40 pcl::PointCloud<pcl::PointXYZRGBA>::Ptr transformedCloudCamA (new pcl:
41 pcl::io::OctreePointCloudCompression<pcl::PointXYZRGBA>* PointCloudEnc
42 pcl::io::OctreePointCloudCompression<pcl::PointXYZRGBA>* PointCloudDec
43 octomap::ColorOcTree rayMap(0.04);
44
45 int nrows = 4;
46 int ncols = 4;
47
48 Eigen::Matrix4f transformCamA = Eigen::Matrix4f::Zero(nrows,ncols);
49 Eigen::Matrix4f transformCamB = Eigen::Matrix4f::Zero(nrows,ncols);
50
51 int positionIndex = 0;
52 bool debug = false;
53 bool rayMapinit = false;
54
55
```

```
56
57 class SimpleOpenNIViewer
58 {
59 public:
60 SimpleOpenNIViewer () : viewer ("PCL␣OpenNI␣Viewer") {}
61 void cloud_cb_2_ (const pcl::PointCloud<pcl::PointXYZRGBA>::ConstPtr &
62 {
63 if (!viewer.wasStopped()){
64 //            auto startPCgenCamB = chrono::steady_clock::now();
65 // Create the filtering object
66 pcl::PassThrough<pcl::PointXYZRGBA> pass;
67
68 // read cloud transformation data from file
69 //            cout << "transformCamB = " << endl << transformCamB << e
70
71
72 pcl::PointCloud<pcl::PointXYZRGBA>::Ptr CloudCamBfiltered (new pcl::Po
73 pcl::PointCloud<pcl::PointXYZRGBA>::Ptr CloudCamBVoxelGrid (new pcl::P
74
75 pass.setInputCloud (cloud);
76 pass.setFilterFieldName ("z");
77 pass.setFilterLimits (0.0, 3.5);
78 //pass.setFilterLimitsNegative (true);
79 pass.filter (*CloudCamBfiltered);
80
81 pcl::VoxelGrid<pcl::PointXYZRGBA> voxelFilter;
82 voxelFilter.setInputCloud(CloudCamBfiltered);
83 voxelFilter.setLeafSize (0.01f, 0.01f, 0.01f);
84 voxelFilter.filter (*CloudCamBVoxelGrid);
85
86 // Executing the transformation
```

```
87  pcl::transformPointCloud (*CloudCamBVoxelGrid, *transformedCloudCamB,
88
89  //    (row, column)
90  viewer.showCloud (transformedCloudCamB, cloudnameB);
91  //            if(debug == true){
92  stringstream stream;
93  stream << "inputCloudCamB.pcd";
94  string filename = stream.str();
95  io::savePCDFileASCII(filename, *CloudCamBfiltered);
96  //            PCL_ERROR("Problem saving %s.\n", filename.c_str());
97  stream.str(std::string());
98  stream << "transformedCloudB.pcd";
99  filename = stream.str();
100 io::savePCDFileASCII(filename, *transformedCloudCamB);
101 stream.str(std::string());
102 stream << "inputCLoudCamB.png";
103 filename = stream.str();
104 io::savePNGFile(filename, *cloud, "rgb");
105 //            cout<< "Saved " << filename << "." << endl;
106 //            cloud = transformedCloudCamB;
107 //            }
108 //            auto endPCgenCamB = chrono::steady_clock::now();
109 //            cout << "Elapsed Time in milliseconds (camB PointCloud g
110 }
111 }
112
113 void cloud_cb_ (const pcl::PointCloud<pcl::PointXYZRGBA>::ConstPtr &cl
114 {
115 if (!viewer.wasStopped()){
116 //            auto startPCgenCamA = chrono::steady_clock::now();
117 // Create the filtering object
```

```
118 pcl::PassThrough<pcl::PointXYZRGBA> pass;
119
120 //            cout << "transformCamA = " << endl << transformCamA << e
121
122 pcl::PointCloud<pcl::PointXYZRGBA>::Ptr CloudCamAfiltered (new pcl::Po
123 pcl::PointCloud<pcl::PointXYZRGBA>::Ptr CloudCamAVoxelGrid (new pcl::P
124
125 pass.setInputCloud (cloud);
126 pass.setFilterFieldName ("z");
127 pass.setFilterLimits (0.0, 4.1);
128 //pass.setFilterLimitsNegative (true);
129 pass.filter (*CloudCamAfiltered);
130
131
132 pcl::VoxelGrid<pcl::PointXYZRGBA> voxelFilter;
133 voxelFilter.setInputCloud(CloudCamAfiltered);
134 voxelFilter.setLeafSize (0.01f, 0.01f, 0.01f);
135 voxelFilter.filter (*CloudCamAVoxelGrid);
136
137
138 // Executing the transformation
139 pcl::transformPointCloud (*CloudCamAVoxelGrid, *transformedCloudCamA,
140
141 //    (row, column)
142 viewer.showCloud (transformedCloudCamA, cloudnameA);
143 //            if(debug == true){
144 stringstream stream;
145 stream << "inputCloudCamA.pcd";
146 string filename = stream.str();
147 io::savePCDFileASCII(filename, *CloudCamAVoxelGrid);
148 //            PCL_ERROR("Problem saving %s.\n", filename.c_str());
```

```
149 stream.str(std::string());
150 stream << "transformedCloudA.pcd";
151 filename = stream.str();
152 io::savePCDFileASCII(filename, *transformedCloudCamA);
153 stream.str(std::string());
154 stream << "inputCLoudCamA.png";
155 filename = stream.str();
156 io::savePNGFile(filename, *cloud, "rgb");
157 //             cout<< "Saved " << filename << "." << endl;
158 //             cloud = transformedCloudCamA;
159 //             }
160 //             auto endPCgenCamA = chrono::steady_clock::now();
161 //            cout << "Elapsed Time in milliseconds (camA PointCloud ge
162 }
163 }
164
165 void run ()
166 {
167 //         //***********************************
168 //         start first Xtion
169 Grabber* interface = new io::OpenNI2Grabber("#1");
170
171 boost::function<void (const pcl::PointCloud<pcl::PointXYZRGBA>::ConstP
172 boost::bind (&SimpleOpenNIViewer::cloud_cb_, this, _1);
173
174 interface->registerCallback (f);
175 interface->start();
176 //***********************************
177
178 //***********************************
179 //         start second Xtion
```

```
180 pcl::Grabber* interface2 = new io::OpenNI2Grabber("#2");
181 boost::function<void (const pcl::PointCloud<pcl::PointXYZRGBA>::ConstP
182 boost::bind (&SimpleOpenNIViewer::cloud_cb_2_, this, _1);
183
184 interface2->registerCallback (f2);
185
186 interface2->start();
187 //*********************************
188
189 while (!viewer.wasStopped())
190 {
191 bool showStatistics = true;
192 bool reset = false;
193
194 pcl::io::compression_Profiles_e compressionProfile = pcl::io::MED_RES_
195
196 PointCloudEncoder = new pcl::io::OctreePointCloudCompression<pcl::Poin
197 PointCloudDecoder = new pcl::io::OctreePointCloudCompression<pcl::Poin
198
199
200 pcl::PointCloud<pcl::PointXYZRGBA>::Ptr concatenatedCloud (new pcl::Po
201 pcl::PointCloud<pcl::PointXYZRGBA>::Ptr cloud_filtered (new pcl::Point
202
203 *concatenatedCloud = *transformedCloudCamA;
204 *concatenatedCloud += *transformedCloudCamB;
205
206 io::savePCDFileASCII("concatenated.pcd", *concatenatedCloud);
207
208 pcl::VoxelGrid<pcl::PointXYZRGBA> voxelFilter;
209 voxelFilter.setInputCloud(concatenatedCloud);
210 voxelFilter.setLeafSize (0.04f, 0.04f, 0.04f);
```

```
211 voxelFilter.filter (*cloud_filtered);
212
213
214 io::savePCDFileASCII("cloud_filtered.pcd", *cloud_filtered);
215
216 // stringstream to store compressed point cloud
217 std::stringstream compressedData;
218
219 // compress point cloud
220 PointCloudEncoder->encodePointCloud (cloud_filtered, compressedData);
221
222 //octomap
223 //            cout<<"copy data into octomap..."<<endl;
224 //
225 //            auto startOctreeGen = chrono::steady_clock::now();
226 octomap::ColorOcTree tree( 0.04 );
227 for (float x = -2.5; x <= 2.5; x += 0.08f) {
228 for (float y = 0; y <= 4; y += 0.08f) {
229 for (float z = -1.5; z <= 2; z += 0.08f) {
230 octomap::point3d endpoint(x, y, z);
231 tree.updateNode(endpoint, false); // integrate 'free' measurement
232 //                          rayMap.updateNode(endpoint, false);
233 }
234 }
235 }
236
237 for (auto p:cloud_filtered->points)
238 {
239 tree.updateNode(octomap::point3d(p.x, p.y, p.z), true);
240 if(rayMapinit == false){
241 rayMap.updateNode(octomap::point3d(p.x, p.y, p.z), true);
```

```
242 }
243 }
244
245
246 for (auto p:cloud_filtered->points)
247 {
248 tree.integrateNodeColor( p.x, p.y, p.z, p.r, p.g, p.b );
249 if(rayMapinit == false){
250 rayMap.integrateNodeColor( p.x, p.y, p.z, p.r, p.g, p.b );
251 }
252 }
253 rayMapinit = true;
254 //            auto endOctreeGen = chrono::steady_clock::now();
255 //            cout << "Elapsed Time in milliseconds (Octomap generatio
256
257 float fixedPos[24][6];
258 //position 1 left side top 1
259 fixedPos[0][0] = -860;
260 fixedPos[0][1] = 1330;
261 fixedPos[0][2] = -560;
262 fixedPos[0][3] = 180;
263 fixedPos[0][4] = 0.0;
264 fixedPos[0][5] = 90.0;
265
266 //position 2 left side bottom
267 fixedPos[1][0] = -860;
268 fixedPos[1][1] = 1330;
269 fixedPos[1][2] = -650;
270 fixedPos[1][3] = 180;
271 fixedPos[1][4] = 0.0;
272 fixedPos[1][5] = 90.0;
```

```
273
274 //position 3 left side top 2
275 fixedPos[2][0] = -860;
276 fixedPos[2][1] = 1330;
277 fixedPos[2][2] = -560;
278 fixedPos[2][3] = 180;
279 fixedPos[2][4] = 0.0;
280 fixedPos[2][5] = 90.0;
281
282 //position 4 right side top 1
283 fixedPos[3][0] = 600;
284 fixedPos[3][1] = 1550;
285 fixedPos[3][2] = -560;
286 fixedPos[3][3] = 180;
287 fixedPos[3][4] = 0.0;
288 fixedPos[3][5] = 90.0;
289
290 //position 5 right side bottom
291 fixedPos[4][0] = 600;
292 fixedPos[4][1] = 1550;
293 fixedPos[4][2] = -655;
294 fixedPos[4][3] = 180;
295 fixedPos[4][4] = 0.0;
296 fixedPos[4][5] = 90.0;
297
298 //position 6 right side top 2
299 fixedPos[5][0] = 600;
300 fixedPos[5][1] = 1550;
301 fixedPos[5][2] = -560;
302 fixedPos[5][3] = 180;
303 fixedPos[5][4] = 0.0;
```

```
304 fixedPos[5][5] = 90.0;
305
306 //position 7 left side top 1
307 fixedPos[6][0] = -860;
308 fixedPos[6][1] = 1550;
309 fixedPos[6][2] = -560;
310 fixedPos[6][3] = 180;
311 fixedPos[6][4] = 0.0;
312 fixedPos[6][5] = 90.0;
313
314 //position 8 left side bottom
315 fixedPos[7][0] = -860;
316 fixedPos[7][1] = 1550;
317 fixedPos[7][2] = -650;
318 fixedPos[7][3] = 180;
319 fixedPos[7][4] = 0.0;
320 fixedPos[7][5] = 90.0;
321
322 //position 9 left side top 2
323 fixedPos[8][0] = -860;
324 fixedPos[8][1] = 1550;
325 fixedPos[8][2] = -560;
326 fixedPos[8][3] = 180;
327 fixedPos[8][4] = 0.0;
328 fixedPos[8][5] = 90.0;
329
330 //position 10 right side top 1
331 fixedPos[9][0] = 600;
332 fixedPos[9][1] = 1330;
333 fixedPos[9][2] = -560;
334 fixedPos[9][3] = 180;
```

```
335 fixedPos[9][4] = 0.0;
336 fixedPos[9][5] = 90.0;
337
338 //position 11 right side bottom
339 fixedPos[10][0] = 600;
340 fixedPos[10][1] = 1330;
341 fixedPos[10][2] = -655;
342 fixedPos[10][3] = 180;
343 fixedPos[10][4] = 0.0;
344 fixedPos[10][5] = 90.0;
345
346 //position 12 right side top 2
347 fixedPos[11][0] = 600;
348 fixedPos[11][1] = 1330;
349 fixedPos[11][2] = -560;
350 fixedPos[11][3] = 180;
351 fixedPos[11][4] = 0.0;
352 fixedPos[11][5] = 90.0;
353
354 //position 13 right side top 1
355 fixedPos[12][0] = 600;
356 fixedPos[12][1] = 1550;
357 fixedPos[12][2] = -560;
358 fixedPos[12][3] = 180;
359 fixedPos[12][4] = 0.0;
360 fixedPos[12][5] = 90.0;
361
362 //position 14 right side bottom
363 fixedPos[13][0] = 600;
364 fixedPos[13][1] = 1550;
365 fixedPos[13][2] = -655;
```

```
366 fixedPos[13][3] = 180;
367 fixedPos[13][4] = 0.0;
368 fixedPos[13][5] = 90.0;
369
370 //position 15 right side top 2
371 fixedPos[14][0] = 600;
372 fixedPos[14][1] = 1550;
373 fixedPos[14][2] = -560;
374 fixedPos[14][3] = 180;
375 fixedPos[14][4] = 0.0;
376 fixedPos[14][5] = 90.0;
377
378 //position 16 left side top 1
379 fixedPos[15][0] = -860;
380 fixedPos[15][1] = 1330;
381 fixedPos[15][2] = -560;
382 fixedPos[15][3] = 180;
383 fixedPos[15][4] = 0.0;
384 fixedPos[15][5] = 90.0;
385
386 //position 17 left side bottom
387 fixedPos[16][0] = -860;
388 fixedPos[16][1] = 1330;
389 fixedPos[16][2] = -650;
390 fixedPos[16][3] = 180;
391 fixedPos[16][4] = 0.0;
392 fixedPos[16][5] = 90.0;
393
394 //position 18 left side top 2
395 fixedPos[17][0] = -860;
396 fixedPos[17][1] = 1330;
```

```
397 fixedPos[17][2] = -560;
398 fixedPos[17][3] = 180;
399 fixedPos[17][4] = 0.0;
400 fixedPos[17][5] = 90.0;
401
402 //position 19 right side top 1
403 fixedPos[18][0] = 600;
404 fixedPos[18][1] = 1330;
405 fixedPos[18][2] = -560;
406 fixedPos[18][3] = 180;
407 fixedPos[18][4] = 0.0;
408 fixedPos[18][5] = 90.0;
409
410 // position 20 right side bottom
411 fixedPos[19][0] = 600;
412 fixedPos[19][1] = 1330;
413 fixedPos[19][2] = -655;
414 fixedPos[19][3] = 180;
415 fixedPos[19][4] = 0.0;
416 fixedPos[19][5] = 90.0;
417
418 //position 21 right side top 2
419 fixedPos[20][0] = 600;
420 fixedPos[20][1] = 1330;
421 fixedPos[20][2] = -560;
422 fixedPos[20][3] = 180;
423 fixedPos[20][4] = 0.0;
424 fixedPos[20][5] = 90.0;
425
426 //position 22 left side top 1
427 fixedPos[21][0] = -860;
```

```
428 fixedPos[21][1] = 1550;
429 fixedPos[21][2] = -560;
430 fixedPos[21][3] = 180;
431 fixedPos[21][4] = 0.0;
432 fixedPos[21][5] = 90.0;
433
434 //position 23 left side bottom
435 fixedPos[22][0] = -860;
436 fixedPos[22][1] = 1550;
437 fixedPos[22][2] = -650;
438 fixedPos[22][3] = 180;
439 fixedPos[22][4] = 0.0;
440 fixedPos[22][5] = 90.0;
441
442 //position 24 left side top 2
443 fixedPos[23][0] = -860;
444 fixedPos[23][1] = 1550;
445 fixedPos[23][2] = -560;
446 fixedPos[23][3] = 180;
447 fixedPos[23][4] = 0.0;
448 fixedPos[23][5] = 90.0;
449
450 float *currentPosition = 0;
451 float originPosition[6];
452 float goalPosition [6];
453 float avoidPositionStart [6];
454 float avoidPositionGoal [6];
455
456 currentPosition = connectReadCartesian(currentPosition);
457 // remove tool from map to avoid collision with itself
458 cout << "**********************************************************
```

```
459 cout << "Delete camera data of robot tool" << endl;
460 cout << "**********************************************************
461 for (float x = (currentPosition[0]/1000)-0.10; x <= (currentPosition[0
462 for (float y = (currentPosition[1]/1000)-0.04; y <= (currentPosition[1
463 for (float z = (currentPosition[2]/1000)-0.04; z <= (currentPosition[2
464 octomap::point3d endpoint(x, y, z);
465 tree.updateNode(endpoint, false); // integrate 'free' measurement
466 //                          rayMap.updateNode(endpoint, false);
467 }
468 }
469 }
470 cout << "**********************************************************
471 cout << "data deleted" << endl;
472 cout << "**********************************************************
473
474 for(int s = 0; s < 6; s++){
475 goalPosition[s] = fixedPos[positionIndex][s];
476 avoidPositionGoal[s] = goalPosition[s];
477 avoidPositionStart[s] =  currentPosition[s];
478 originPosition[s] = currentPosition[s];
479 }
480 if(debug == true){
481 cout << "**********************************************************
482 cout << "Initialisation" << endl;
483 cout << "**********************************************************
484 printf("avoidPositionStart X pos: %f \n", avoidPositionStart[0]);
485 printf("avoidPositionStart Y pos: %f \n", avoidPositionStart[1]);
486 printf("avoidPositionStart Z pos: %f \n", avoidPositionStart[2]);
487 printf("avoidPositionStart W pos: %f \n", avoidPositionStart[3]);
488 printf("avoidPositionStart P pos: %f \n", avoidPositionStart[4]);
489 printf("avoidPositionStart R pos: %f \n", avoidPositionStart[5]);
```

```
490 cout << "**********************************************************
491 cout << "**********************************************************
492 printf("avoidPositionGoal X pos: %f \n", avoidPositionGoal[0]);
493 printf("avoidPositionGoal Y pos: %f \n", avoidPositionGoal[1]);
494 printf("avoidPositionGoal Z pos: %f \n", avoidPositionGoal[2]);
495 printf("avoidPositionGoal W pos: %f \n", avoidPositionGoal[3]);
496 printf("avoidPositionGoal P pos: %f \n", avoidPositionGoal[4]);
497 printf("avoidPositionGoal R pos: %f \n", avoidPositionGoal[5]);
498 cout << "**********************************************************
499 cout << "Initialisation" << endl;
500 cout << "**********************************************************
501 }
502 bool pathFound = false;
503 int avoidAttemptCnt = 1;
504
505 tree.updateInnerOccupancy();
506 tree.write( "output_file.ot" );
507
508 while(pathFound == false && reset == false){
509 cout << "**********************************************************
510 cout << "PositionIndex #" << positionIndex << endl;
511 cout << "**********************************************************
512 cout << "Attempt #" << avoidAttemptCnt << " to move" << endl;
513
514 octomap::point3d rayStart(avoidPositionStart[0]/1000, avoidPositionSta
515 octomap::point3d rayDirection((avoidPositionGoal[0]-avoidPositionStart
516 octomap::point3d end;
517 octomap::point3d rayEnd(avoidPositionGoal[0]/1000, avoidPositionGoal[1
518 double rayDistance = sqrt(pow((avoidPositionGoal[0]-avoidPositionStart
519
520 bool obstacleFound = tree.castRay(rayStart, rayDirection, end, true, r
```

```
521
522 cout << "rayDirection␣x:" << rayDirection.x() << "␣␣␣y:" << rayDirecti
523 std::vector<octomap::point3d> ray;
524 //                if(debug == true){
525 rayMap.computeRay(rayStart, rayEnd, ray);
526 cout << "after␣computeRay" << endl;
527 //                cout << "ray.size() ==" << ray.size()<< endl;
528
529 // compute Ray for rayMap
530
531 for(unsigned int x = 0; x < ray.size(); x++){
532
533 rayMap.updateNode( octomap::point3d(ray[x].x(), ray[x].y(), ray[x].z()
534 rayMap.integrateNodeColor(ray[x].x(), ray[x].y(), ray[x].z(), 255, 0,
535 //                        cout << "rayCoordinates #" << x <<" :
    x:" << ray[x].x() <<"   y:" << ray[x].y()<< "   z:" << ray[x].z() << e
536 }
537
538
539 rayMap.updateInnerOccupancy();
540 rayMap.write( "rayMap.ot" );
541 cout << "after␣rayMap" << endl;
542 //                }
543
544 // Vertical Movement not regarded for collision avoidance
545 if(positionIndex == 2 || positionIndex == 5 || positionIndex == 8 || p
    // maybe directional condition based on rayDirection to check if robot
546 if(debug == true){do {
547 cout << "**************************************************" << endl;
548 cout << "moving␣to␣upper␣position,␣press␣any␣key␣to␣continue" << endl;
549 cout << "**************************************************" << endl;
```

```
550
551 } while (cin.get() != '\n');
552 }
553 connectWriteCartesian(avoidPositionGoal, currentPosition);
554 positionIndex ++;
555 pathFound = true;
556 }
557 // Vertical Movement not regarded for collision avoidance
558 else if(positionIndex == 1 || positionIndex == 7 || positionIndex == 1
    // maybe directional condition based on rayDirection to check if robot
559 if(debug == true){
560 do {
561 cout << "*****************************************************" << en
562 cout << "moving to gripping position, press any key to continue" << en
563 cout << "*****************************************************" << en
564
565 } while (cin.get() != '\n');
566 }
567 connectWriteCartesian(avoidPositionGoal, currentPosition);
568 sleep(2);
569 gripperOn();
570 positionIndex ++;
571 pathFound = true;
572 }
573 // Vertical Movement not regarded for collision avoidance
574 else if(positionIndex == 4  || positionIndex == 10 || positionIndex ==
    // maybe directional condition based on rayDirection to check if robot
575 if(debug == true){
576 do {
577 cout << "*****************************************************" << en
578 cout << "moving to gripping position, press any key to continue" << en
```

```
579 cout << "*****************************************************" << en
580
581 } while (cin.get() != '\n');
582 }
583 connectWriteCartesian(avoidPositionGoal, currentPosition);
584 sleep(2);
585 gripperOff();
586 positionIndex ++;
587 pathFound = true;
588 }
589
590 //Raycasting not finding any object in the path
591 else if( obstacleFound == false){
592 if(debug == true){
593 do {
594 cout << "*************************************************" << endl;
595 cout << "no obstacle, press any key to move to goalPosition" << endl;
596 cout << "*************************************************" << endl;
597
598 } while (cin.get() != '\n');
599 }
600 connectWriteCartesian(avoidPositionGoal, currentPosition);
601 if(avoidPositionGoal != goalPosition){
602 cout << "avoidPositionGoal != goalPosition" << endl;
603 currentPosition = connectReadCartesian(currentPosition);
604 connectWriteCartesian(goalPosition, currentPosition);
605 }
606 avoidAttemptCnt = 1;
607 positionIndex ++;
608 pathFound = true;
609 }
```

```
610
611 //Raycasting finding any object in the path
612 else if(obstacleFound == true){
613 //                    auto startAvoid = chrono::steady_clock::now();
614 // Attempts to avoid collisions too high!
615 if(avoidAttemptCnt >= 6){
616
617 do {
618 cout << "*************************************************************
619 cout << "can't␣find␣a␣path,␣remove␣object␣and␣press␣any␣key␣to␣move␣to
620 cout << "␣␣␣␣␣␣␣␣origin␣position␣and␣restart␣collision␣detection" << en
621 cout << "*************************************************************
622
623 } while (cin.get() != '\n');
624
625 connectWriteCartesian(originPosition, currentPosition);
626 //                        currentPosition = connectReadCartesian(curre
627 for(int s = 0; s < 6; s++){
628 avoidPositionGoal[s] = goalPosition[s];
629 //                             avoidPositionStart[s] =  currentPosition
630 }
631 avoidAttemptCnt = 1;
632 reset = true;
633 }
634 // Attempting to avoid collision by moving in z-Direction
635 else{
636 if(debug == true){
637 do {
638 cout << "*************************************************************
639 cout << "obstacle␣found,␣moving␣up␣to␣avoid␣collision.␣Press␣any␣key␣t
640 cout << "*************************************************************
```

```
641
642 } while (cin.get() != '\n');
643 }
644 avoidPositionStart[2] += 100;
645 avoidPositionGoal[2] +=100;
646 if(debug == true){
647 cout << "*************************************************************
648 cout << "After z increment" << endl;
649 cout << "*************************************************************
650 printf("avoidPositionStart X pos: %f \n", avoidPositionStart[0]);
651 printf("avoidPositionStart Y pos: %f \n", avoidPositionStart[1]);
652 printf("avoidPositionStart Z pos: %f \n", avoidPositionStart[2]);
653 printf("avoidPositionStart W pos: %f \n", avoidPositionStart[3]);
654 printf("avoidPositionStart P pos: %f \n", avoidPositionStart[4]);
655 printf("avoidPositionStart R pos: %f \n", avoidPositionStart[5]);
656 cout << "*************************************************************
657 cout << "After z increment" << endl;
658 cout << "*************************************************************
659 }
660 connectWriteCartesian(avoidPositionStart, currentPosition);
661 currentPosition = connectReadCartesian(currentPosition);
662 avoidAttemptCnt ++;
663 }
664 //                    auto endAvoid = chrono::steady_clock::now();
665 //                    cout << "Elapsed Time in milliseconds (avoid obs
666 }
667
668 // counter for fixed positions reset to start from the beginning.
669 if(positionIndex >= 24){
670 positionIndex = 0;
671 }
```

```
672 if(debug == true){
673 do {
674 cout << "*****************************************" << endl;
675 cout << "checkpoint reached, press any key to continue" << endl;
676 cout << "*****************************************" << endl;
677 } while (cin.get() != '\n');
678 }
679 }
680 }
681 interface->stop ();
682 interface2->stop ();
683 }
684
685 pcl::visualization::CloudViewer viewer;
686 const std::string cloudnameA = "cloudA";
687 const std::string cloudnameB = "cloudB";
688 };
689
690 int main (int argc, char * const argv[])
691 {
692 if(argc > 1){
693 if (argv[1] == std::string("-d")){
694 cout << "*****************************************" << endl;
695 cout << "Entering Debug Mode, have fun pushing buttons" << endl;
696 cout << "*****************************************" << endl;
697 debug = true;
698 }
699 }
700
701 // read cloud transformation data from file
702 ifstream fin ("./../Rob195/cloudA.txt");
```

```
703
704 if (fin.is_open())
705 {
706 for (int row = 0; row < nrows; row++)
707 for (int col = 0; col < ncols; col++)
708 {
709 float item = 0.0;
710 fin >> item;
711 transformCamA(row, col) = item;
712 }
713 fin.close();
714 }
715 ifstream fin2 ("./../Rob195/cloudB.txt");
716 {
717 for (int row = 0; row < nrows; row++)
718 for (int col = 0; col < ncols; col++)
719 {
720 float item = 0.0;
721 fin2 >> item;
722 transformCamB(row, col) = item;
723 }
724 fin2.close();
725 }
726
727 SimpleOpenNIViewer v;
728 v.run ();
729
730 return 0;
731 }
```

H. camCalibration main.cp code

```
1  #include <sys/socket.h>
2  #include <netinet/in.h>
3  #include <arpa/inet.h>
4  #include <cstring>
5  #include <iostream>
6  #include <stdlib.h>
7  #include <string>
8  #include <math.h>
9  #include <chrono>
10
11 #include <pcl/io/openni2_grabber.h>
12 #include <pcl/visualization/cloud_viewer.h>
13 #include <pcl/io/pcd_io.h>
14 #include <pcl/io/png_io.h>
15 #include <pcl/io/ply_io.h>
16 #include <pcl/point_types.h>
17 #include <pcl/console/parse.h>
18 #include <pcl/common/transforms.h>
19 #include <pcl/point_types.h>
20 #include <pcl/filters/passthrough.h>
21 #include <pcl/compression/octree_pointcloud_compression.h>
22 #include <pcl/filters/voxel_grid.h>
23 bool camAsaved = false;
24 bool camBsaved = false;
```

```
25
26
27 using namespace std;
28 using namespace pcl;
29
30
31 class SimpleOpenNIViewer
32 {
33 public:
34 SimpleOpenNIViewer () : viewer ("PCL␣OpenNI␣Viewer") {}
35 void cloud_cb_2_ (const pcl::PointCloud<pcl::PointXYZRGBA>::ConstPtr &
36 {
37 if (!viewer.wasStopped()){
38
39 pcl::PassThrough<pcl::PointXYZRGBA> pass;
40
41 pcl::PointCloud<pcl::PointXYZRGBA>::Ptr CloudCamBfiltered (new pcl::Po
42
43 pass.setInputCloud (cloud);
44 pass.setFilterFieldName ("z");
45 pass.setFilterLimits (0.0, 3.5);
46 //pass.setFilterLimitsNegative (true);
47 pass.filter (*CloudCamBfiltered);
48
49
50 viewer.showCloud (CloudCamBfiltered, cloudnameB);
51 stringstream stream;
52 stream << "referenceCloudB.pcd";
53 string filename = stream.str();
54 io::savePCDFileASCII(filename, *CloudCamBfiltered);
55
```

```
56 stream.str(std::string());
57 stream << "referenceCloudB.png";
58 filename = stream.str();
59 io::savePNGFile(filename, *CloudCamBfiltered, "rgb");
60 }
61 }
62
63 void cloud_cb_ (const pcl::PointCloud<pcl::PointXYZRGBA>::ConstPtr &cl
64 {
65 if (!viewer.wasStopped()){
66 pcl::PassThrough<pcl::PointXYZRGBA> pass;
67
68 pcl::PointCloud<pcl::PointXYZRGBA>::Ptr CloudCamAfiltered (new pcl::Po
69
70 pass.setInputCloud (cloud);
71 pass.setFilterFieldName ("z");
72 pass.setFilterLimits (0.0, 4.1);
73 //pass.setFilterLimitsNegative (true);
74 pass.filter (*CloudCamAfiltered);
75
76 viewer.showCloud (CloudCamAfiltered, cloudnameA);
77 stringstream stream;
78 stream << "referenceCloudA.pcd";
79 string filename = stream.str();
80 io::savePCDFileASCII(filename, *CloudCamAfiltered);
81
82 stream.str(std::string());
83 stream << "referenceCloudA.png";
84 filename = stream.str();
85 io::savePNGFile(filename, *CloudCamAfiltered, "rgb");
86 }
```

```
87  }
88
89  void run ()
90  {
91  //         //#################################
92  //         start first Xtion
93  Grabber* interface = new io::OpenNI2Grabber("#1");
94
95  boost::function<void (const pcl::PointCloud<pcl::PointXYZRGBA>::ConstP
96  boost::bind (&SimpleOpenNIViewer::cloud_cb_, this, _1);
97
98  interface->registerCallback (f);
99  interface->start();
100
101
102 //#################################
103 //#################################
104 //         start second Xtion
105 Grabber* interface2 = new io::OpenNI2Grabber("#2");
106 boost::function<void (const pcl::PointCloud<pcl::PointXYZRGBA>::ConstP
107 boost::bind (&SimpleOpenNIViewer::cloud_cb_2_, this, _1);
108
109 interface2->registerCallback (f2);
110 interface2->start();
111 //#################################
112
113 while (!viewer.wasStopped())
114 {
115
116 }
117 interface->stop ();
```

```
118 interface2->stop ();
119 }
120
121 pcl::visualization::CloudViewer viewer;
122 const std::string cloudnameA = "cloudA";
123 const std::string cloudnameB = "cloudB";
124 };
125
126
127 int main ()
128 {
129
130 int tableLength = 2250;
131 int tableWidth  = 1480;
132 float tableHeight = 670;
133
134 float tableStartingpointX = -1274.883;
135 float tableStartingpointY = 1154.362;
136 float tableStartingpointZ = -653.149;
137
138 // Fill in the cloud data
139
140 pcl::PointCloud<pcl::PointXYZRGBA>::Ptr basic_cloud_ptr (new pcl::Poin
141 uint8_t r(255), g(15), b(15);
142 uint8_t r3(15), g3(15), b3(255);
143 for (float xAxis = tableStartingpointX; xAxis < tableStartingpointX+ta
144 for(float yAxis = tableStartingpointY; yAxis < tableStartingpointY+tab
145 //for(float zAxis = tableStartingpointZ; zAxis < tableStartingpointZ+t
146 pcl::PointXYZRGBA basic_point;
147 basic_point.x = xAxis/1000;
148 basic_point.y = yAxis/1000;
```

```
149 basic_point.z = tableStartingpointZ/1000;
150
151
152 if(xAxis <= tableStartingpointX+1000){
153 if(yAxis <= tableStartingpointY+1000){
154 uint32_t rgb = (static_cast<uint32_t>(r3) << 16 |
155 static_cast<uint32_t>(g3) << 8 | static_cast<uint32_t>(b3));
156 basic_point.rgb = *reinterpret_cast<float*>(&rgb);
157 }
158 else{
159 uint32_t rgb = (static_cast<uint32_t>(r) << 16 |
160 static_cast<uint32_t>(g) << 8 | static_cast<uint32_t>(b));
161 basic_point.rgb = *reinterpret_cast<float*>(&rgb);
162 }
163 }
164 else if(xAxis > (tableStartingpointX+tableLength) -1000){
165 if(yAxis > (tableStartingpointY+tableWidth)-1000){
166 uint32_t rgb = (static_cast<uint32_t>(r3) << 16 |
167 static_cast<uint32_t>(g3) << 8 | static_cast<uint32_t>(b3));
168 basic_point.rgb = *reinterpret_cast<float*>(&rgb);
169 }
170 else {
171 uint32_t rgb = (static_cast<uint32_t>(r) << 16 |
172 static_cast<uint32_t>(g) << 8 | static_cast<uint32_t>(b));
173 basic_point.rgb = *reinterpret_cast<float*>(&rgb);
174 }
175
176 }
177 else{
178 uint32_t rgb = (static_cast<uint32_t>(r) << 16 |
179 static_cast<uint32_t>(g) << 8 | static_cast<uint32_t>(b));
```

```
180 basic_point.rgb = *reinterpret_cast<float*>(&rgb);
181 }
182
183
184 basic_cloud_ptr->points.push_back(basic_point);
185
186
187 //}
188 }
189 }
190
191
192 uint8_t r2(15), g2(255), b2(15);
193 for (float xAxis = tableStartingpointX; xAxis < tableStartingpointX+50
194 for(float yAxis = tableStartingpointY; yAxis < tableStartingpointY+500
195 //for(float zAxis = tableStartingpointZ; zAxis < tableStartingpointZ+t
196 pcl::PointXYZRGBA basic_point;
197 basic_point.x = xAxis/1000;
198 basic_point.y = yAxis/1000;
199 basic_point.z = (tableStartingpointZ - 100)/1000;
200 uint32_t rgb = (static_cast<uint32_t>(r2) << 16 |
201 static_cast<uint32_t>(g2) << 8 | static_cast<uint32_t>(b2));
202 basic_point.rgb = *reinterpret_cast<float*>(&rgb);
203 basic_cloud_ptr->points.push_back(basic_point);
204
205 //}
206 }
207 }
208
209 basic_cloud_ptr->width = (int) basic_cloud_ptr->points.size ();
210 basic_cloud_ptr->height = 1;
```

```
211
212 pcl::io::savePCDFileASCII ("referenceCloudTable.pcd", *basic_cloud_ptr
213 std::cerr << "Saved␣data␣points␣to␣referenceCloudTable.pcd." << std::e
214
215 SimpleOpenNIViewer v;
216 v.run ();
217
218 return (0);
219 }
```