Technische Universität Dresden
Faculty of Electrical and Computer Engineering
Institute of Communication Technology
Deutsche Telekom Chair of Communication Networks

Oberseminar + PBL

Reducing the Traffic in the Control Loop for the Robot Arm

Shen, Yunbin
4724075
China

Supervisors:
M.Sc. Elena Urbano Pérez
Dr.-Ing. Ievgenii Tsokalo

Dresden, 27.02.2019

Declaration

I declare that I have written the present work independently and have used no other than the specified sources and resources. I submit it for the first time as an exam. I am aware that an attempted fraud will be punished with the grade "not sufficient" (5.0) and in case of recurrence may lead to exclusion from the performance of further examinations.

Surname: Shen

First Name: Yunbin

Martrikel-Nr: 4724075

Dresden, _____ Signature: _____

Task description

Topic 4 “Reducing the traffic in the control loop for the robot arm”

In default mode, the robot sends a bulky description of its state to the workstation. The application running on the workstation rarely needs all state information. As a result, the most part of transferred data is not used.

The capacity of the communication channel can be spared by transferring only application relevant data. For this purpose, the interface should be added that filter the information between the robot and the workstation.

The student will transform the basic libfranka example with linear Cartesian movement by adding this interface. In the basic example, the robot communicates directly to the application running on the workstation. The interface should be located between the robot and the application acting as a middle man. It filters the information in both directions. The state information coming from robot is inspected and only the relevant data is forwarded to the application. In backward direction, the commands arriving from the application are transformed to the format know by the robot.

In this work, the application and the interface run on the same PC, the workstation. The communication between the application and the interface shall be organized through the UDP socket on the localhost.

Abstract

This project is to create a interface between the robot and the workstation to reduce the traffic in the control loop for the robot arm. In default mode, the robot sends a bulky description of its state to the workstation. Meanwhile, a lot of data is not used by the application. Therefore, it is necessarily to reduce the traffic between the robot and workstation.

This interface program is based on C++ and UDP socket communication. It include 6 functions, i.e. data extraction and restoring, UDP communication, Compression and CRC32 checking and UDP communication, all of these function are designed to reduce the traffic. Firstly, with the data extraction and restoring, it can sends the information flexibly between the robot and workstation to cut down the traffic. Secondly, the RLE Compression provide a very fast and effective way to reduce the amount of sent data. Thirdly, several UDP communication methods are introduced, which are designed in order to reduce the time consumption of the UDP communication in the robot control loop. Meanwhile, there also some problems with the UDP communications, and the reasons are talked in the section 3.

Keywords: Franka, Traffic reducing, Interface, C++, UDP, Data Filtering, Compression

Contents

List of Abbreviations

CRC	Cyclic redundancy check
UDP	User Datagram Protocol
RLE	Run-length Encoding
FCI	Franka Control Interface

1. Introduction

1.1. Franka Robot

The Franka Control Interface (FCI) allows a fast and direct low-level bidirectional connection to the Arm and Hand [1]. It can provides the current status of the robot and enables a directly control of the robot arm.

The Hardware of the Franka robot consists of workstation computer, controller and robot arm. The workstation PC and the controller are connected by LAN. With the open source c++ library "libfranka", the workstation can send real-time control values at 1 kHz [2]. Because of the 1 kHz control frequency, it is important to limit the latency of the workstation PC.

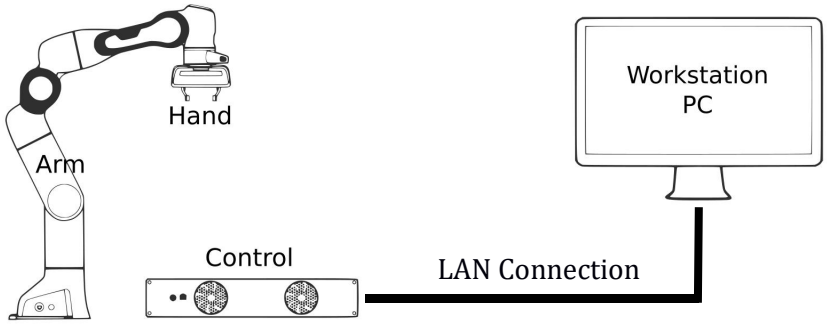


Figure 1.1.1 Franka Robot Arm [1]

1.2. The class franka::RobotState

To storage the control values and the current robot states, the class franka::RobotState is used in the Franka Control Interface. It storage the cartesian pose, the cartesian velocities, the joint pose, the joint velocities and so on. Meanwhile the errors and the duration are also inclusive in the class franka::RobotState. All the data sets in the class franka::RobotState is listed in the Figure 1.2.1.

NO.	NAME	SIZE(BYTE)	NO.	NAME	SIZE(BYTE)
1	O_T_EE	128	23	q_d	56
2	O_T_EE_d	128	24	dq	56
3	F_T_EE	128	25	dq_d	56
4	EE_T_K	128	26	ddq_d	56
5	m_ee	8	27	joint_contact	56
6	I_ee	72	28	cartesian_contact	48
7	F_x_Cee	24	29	joint_collision	56
8	m_load	8	30	cartesian_collision	48
9	I_load	72	31	tau_ext_hat_filtered	56
10	F_x_Cload	24	32	O_F_ext_hat_K	48
11	m_total	8	33	K_F_ext_hat_K	48
12	I_total	72	34	O_dP_EE_d	48
13	F_x_Ctotal	24	35	O_T_EE_c	128
14	elbow	16	36	O_dP_EE_c	48
15	elbow_d	16	37	O_ddP_EE_c	48
16	elbow_c	16	38	theta	56
17	delbow_c	16	39	dtheta	56
18	ddelbow_c	16	40	current_errors	37
19	tau_J	56	41	last_motion_errors	37
20	tau_J_d	56	42	control_command_success_rate	8
21	dtau_J	56	43	robot_mode	1
22	q	56	44	time	8

Figure 1.2.1 The members of the class franka::RobotState

Therefore, there are a bulky information in the franka::RobotState, while the most descriptions are useless. For controlling the robot, there are only totally 4 methods, i.e. Joint Positions, Joint Velocity, Cartesian Pose, Cartesian Velocities. And the state, which are used for generate the motion are only the state “O_T_EE_c”, which is the Last commanded end effector pose of motion generation in base frame, and the sate “q_d”, which is the desired joint position [3]. And these two states can be replaced with each other, because these are only two different descriptions of the same pose of the robot arm. In the example “generate_cartesian_pose_motion”, there is only one member i.e. “O_T_EE_c” used. Therefore, in the most cases, there are only one or several states in the franka::RobotState are used. So it is important to extract the useful information from the franka::RobotState to reducing the traffic.

1.3. UDP Communication

UDP is the short of User Datagram Protocol. It was designed by David P. Reed in 1980 and formally defined in RFC 768. Unlike TCP, there is no 3-Way Handshake and no guarantee of that

the sent data is correct or not. So the latency of the UDP is much smaller than TCP. Therefore, the UDP communication is very suitable for time-sensitive applications [4].

1.4. Run-length Encoding Compression

Run-length encoding (RLE) is a very simple form of lossless data compression. It is suitable for the situations where there are many duplicate data like simple graphic images. But in some situations it could greatly increase the file size [5].

In this project, The mostly used data type in franka::robot_state is double-precision floating-point numbers. Most of the members in the franka::robot_state is a n-double-precision floating-point number array. The structure of a double-precision floating-point number is (Figure1.4.1)

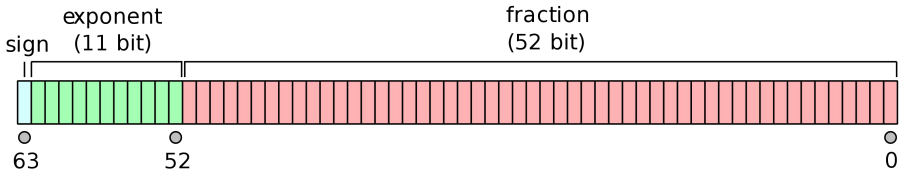


Figure 1.4.1 Double-precision floating-point format [6]

When some number in the array is not used, the value of these numbers are always 0. Then the corresponding memory format is 0x 0000 0000 (Figure 1.4.2).

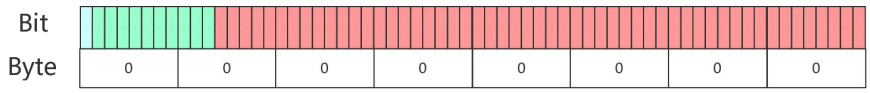


Figure 1.4.2 The Memory format of double-precision floating-point number 0

Therefore, there are sometimes many consecutive and useless “0” in the char array to be sent. So it is very effective to compress these 0 to reduce the size. With the compression, the traffic can also be reduced with the control accuracy decreases.

2. The Robot Communication Interface

In this chapter, it will introduce the structure of the whole interface and describe the implementation of each feature, the performances and the test result.

2.1. The Package Robot Communication Interface

The following interface and the application programs are mainly based on the package robot communication interface. This package provide all needed features for the application and interface programs. It contains 6 parts (Figure2.1.1), the main class “Data Service” and the classes of String Process, Data Process, Data Interface, UDP, CRC32 and Compression. The features data extraction and restoring are integrated in the main class Data Process. And the features like CRC32, Compression can be enabled or disabled according to the usage.

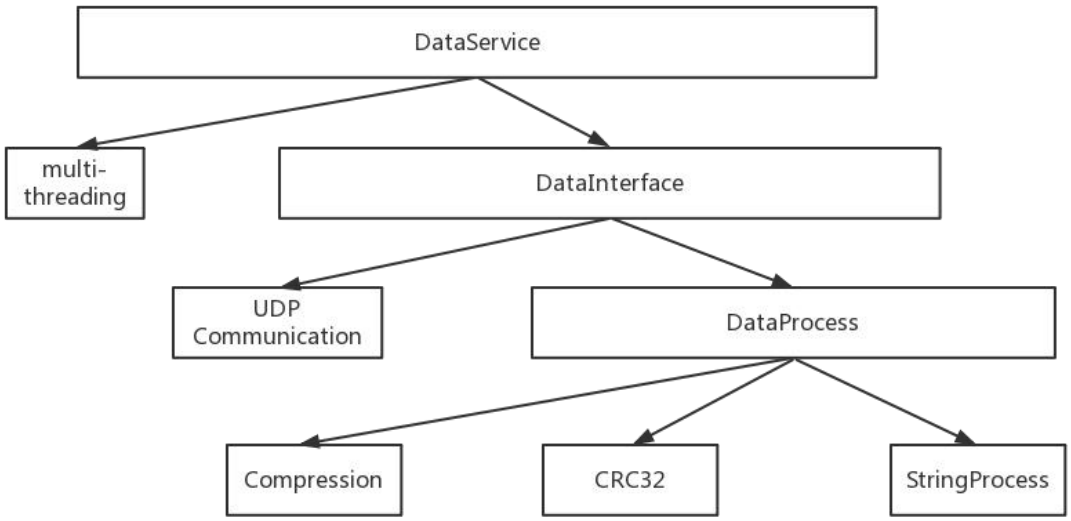


Figure 2.1.1 Class diagram of the package robot communication interface

2.2. Data Extraction and Restoring

The principle of the data extraction in this program is to extract the bytes from the specific locations in the storage space of the franka::RobotState, i.e, the program convert the needed data sets of the franka::RobotState to a char array. The conversion of franka::robot_state to char array is very simple, the code is,

```
template <class T>
uchar8t *DataToStr(T &robot_state, uchar8t *&str)
{
    return mStrCpy(str, sizeof(robot_state), (uchar8t *)(&robot_state));
}
```

Where the function mStrCpy copy the char array (uchar8t *)(&robot_state) to the char array str. And the "robot_state" in the program can be replaced by any member in the class franka::Robot_state.

As for data restoring, the program can according to the location of each data set in the received char array to convert the bytes to a specific data type. It is very simple to use pointer to convert the received data to a normal franka::robot_state. The code is,

```
template <class T>
uchar8t *StrToData(uchar8t *&str, T &data)
{
    data = *(T *)(str);
    return str += sizeof(data);
}
```

Where the unsigned char pointer *str is the location of the determinate data set in the received char array.

The flow chart of the data extraction and restoring for the example "generate_cartesian_pose_motion" is shown in Figure 2.2.1.

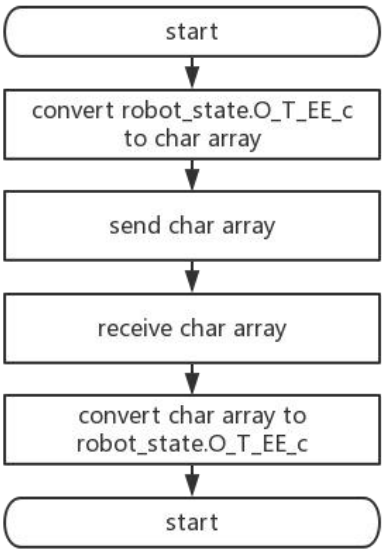


Figure 2.2.1 extraction and restoring of robot_state.O_T_EE_c.

2.3. Data Compression

2.3.1. Introduce

The Compression algorithm used in this program is a special RLE compression. As shown in Figure 2.3.1 the program first detects consecutive zeros and then converts them to a form like "0N", where the byte "N" represents the number of zeros.

Figure 2.3.1 The special RLE compression

Because that the data length of the franka::robot_state is more than 2000 byte, it is possible that there are more than 256 bytes consecutive zeros to be compressed. So it may cause errors when only 1 byte N to represent the number of zeros. Meanwhile, using 2 bytes to represent the number of zeros is too expansive because in most cases there are not so many consecutive zeros. Therefore, the first 1 bit of the byte N is used to indicate that there are 1 or 2 bytes to represent the number of zeros. When the number of zeros is less than 128, then there is only one byte N, the value is the same as number of zeros. When the number is more than 128, then there are 2

bytes i.e. N_1 N_2 represent the number of zeros (Figure 2.3.2), the value of the N_1, N_2 are

$$N_1 = (N\%128)\,|\,0x80 \qquad N_2 = N/128$$

Where N is the number of zeros. At the same time the value of the first bit of N_1 is 1 in this case.

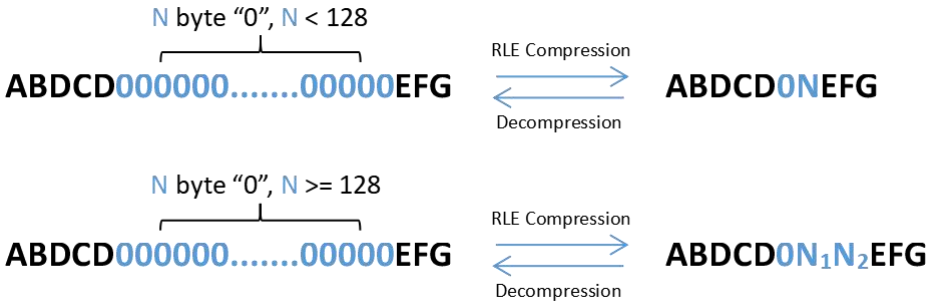


Figure 2.3.2 The special RLE compression used in program

The code of compression is

```
void RleCompress(const uchar8t *src, const ushort16t length, uchar8t *dst_start, ushort16t &new_length)
{
    uchar8t *dst = dst_start;
    uint32t count = 0;
    dst += 2;
    for (int32t i = 0; i < length; i++)
    {
        if (*src == 0)
        {
            src++;
            count++;
            continue;
        }
        if (count != 0)
        {
            *dst++ = 0;
            if (count < 128)
            {
                *dst++ = count;
            }
            else
            {
                *dst++ = uchar8t(count % 128) | 0x80;
                *dst++ = uchar8t(count / 128);
            }
            count = 0;
        }
        *dst++ = *src++;
```

```
        }
        if (count != 0)
        {
            *dst++ = 0;
            if (count < 128)
            {
                *dst = count;
            }
            else
            {
                *dst++ = uchar8t(count % 128) | 0x80;
                *dst++ = uchar8t(count / 128);
            }
        }
        new_length = dst - dst_start + 1;
        *(ushort16t *)(dst_start) = (ushort16t)(new_length);
    }
```

The code of Decompression is

```
    void RleDecompress(const uchar8t *src, uchar8t *dst_start)
    {
        ushort16t length = *(ushort16t *)(src);
        uchar8t *dst = dst_start;
        ushort16t count = 0;
        src += 2;
        for (int32t i = 2; i < length; i++)
        {
            if (*src == 0)
            {
                src++;
                count = ushort16t(*(src));
                if ((*src) & 0x80)
                    count = count - 0x80 + ushort16t(*++src) * 0x80;
                while (count--)
                    *dst++ = 0;
                src++;
                continue;
            }
            *dst++ = *src++;
        }
    }
```

2.3.2. The Pros and cons

Because the function and calculation of this special RLE Compression are very simple, so its

compression and decompression speed are much faster than any other compression methods. The disadvantage is that this method is just suitable for special situation, like in the example, there are only two numbers in the robot_state.O_T_EE_c (a 16-double-precision floating-point number array) are used. When it is used for other normal situations, its performance will be very terrible.

2.3.3. The Performance and Results

The tested data is the values in the robot_state.O_T_EE_c, which is used in the example "generate_cartesian_pose_motion". According to the example program, the values of the robot_state.O_T_EE_c are set like below

```
robot_state.O_T_EE_c = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -0.999023, 0, -0.00810967, 0, 0};
```

And the performance of the RLE Compression has been compared with quicklz [7], snappy [8], zlib [9], which are very famous compression methods and widely used. The result is shown in Figure 2.3.3.

Compress method		Quicklz	Zlib	Snappy	RLE
Size (bytes)	Original	128			
	Compressed	41	33	41	24
Time (us)		101	203	26	1

Figure 2.3.3 The Performances of different Compression

The performance is very perfect, i.e. the RLE Compression has the smallest compressed size and uses only 1us, which is much faster than any other methods.

2.4. The first synchronous UDP Communication

2.4.1. The Structure of the UDP Communication

The UDP communication programs runs on both the application and the interface device, which is located between the robot and the application. In this first communication method the Interface device works as a server, therefore, there is not need to set the application IP address, and it will be easier to setup the connection of application and the interface device.

On the application side the program works as a client, its task is to cut down the data quantity and send reliably to the server, the interface program, which can directly connect the robot through the cable. Then, the server will convert the received data to a normal franka::robot_state, which can direct used by robot. The structure of the whole system is shown in the Figure2.4.1.

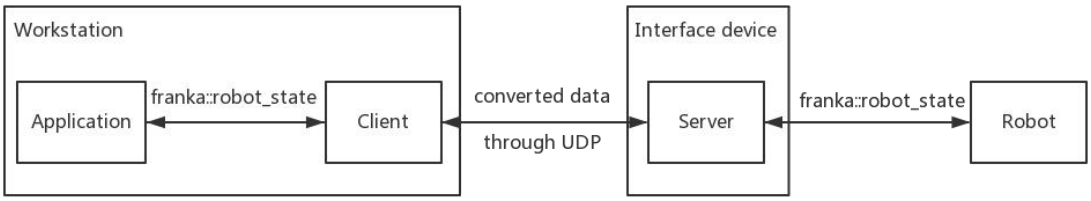


Figure 2.4.1 The structure of the whole system

The Figure2.4.2 shows the flow chart of the application (client) program. At first, the program need to be told that which data set of the franka::robot_state is sent through the UDP socket. Then, the program will extract the specific data sets from the franka::robot_state. After that, the program begin to generate the crc32 check code and compress the data. Finally, the compressed data is sent through the UDP socket to the server.

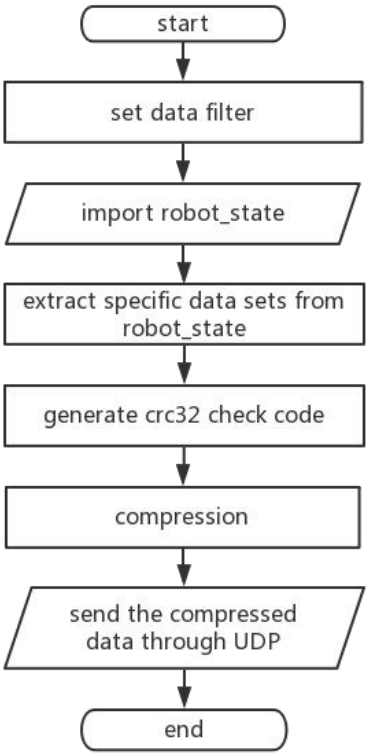


Figure 2.4.2 The flow chart of the application (client) program

The Figure2.4.3 shows the flow chart of the interface (server) program. The program receive the data from the application (client) through UDP socket, after decompression and CRC32 checking, the data the received data will be converted to a normal franka::robot_state. Then, the interface program send the control values to the robot directly .

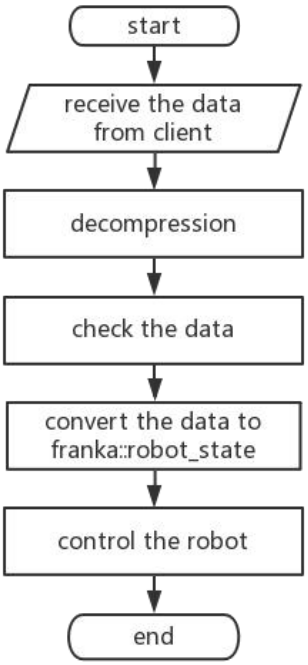


Figure 2.4.3 The flow chart of the interface (server) program

2.4.2. The Pros and cons

The advantages of this UDP communication method is that the quantity of the sent data can be very small with compression and data filtering. Then, the CRC32 check can guarantee the data reliability. But the disadvantages are also very obviously, all of these functions costs some time to calculate, and may not suitable for poor hardware.

2.4.3. The Performance and Results

When this program ran with the robot, the robot aborted because of the communication rate was too slow. To find out the reason, I have tested the time consumption in the robot control

loop like in Figure 2.4.2. The time of receiving is always more than 400us, and the time of sending the message is between 32-200us.

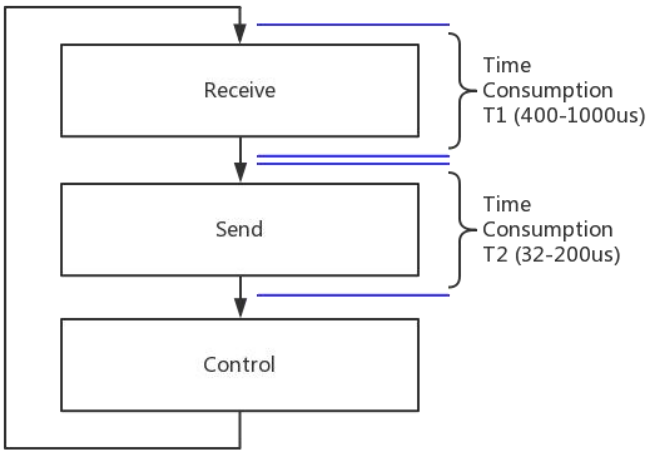


Figure 2.4.4

According to the test result. The reason of abortion may firstly caused by that the program used a lot of time because of crc32 check and compression, which should be used to make the data reliably and reduce the time of communication through UDP, but increase the time consumption of the calculation in the program. Then, the UDP communication is synchronous, as a result, the data receiver can block the entire thread and wait for the message, which costs a lot of time. So my supervisors have told me in the next step to change the program to a asynchronous UDP communication.

2.5. The asynchronous UDP Communication -1

To solve the time out problem, a asynchronous UDP communication method is used in the interface program. The UDP communication part is base on C and run on a another threads in the program, like Xia, Funing's C++ boost asynchronous UDP communication example, which runs totally fine and helps me a lot. Additionally, a thread block is used for data transfer between different threads. To make the communication as fast as possible, the compression and crc32 check are disabled in the following cases.

2.5.1. The Structure of the UDP Communication

The interface program works still as a UDP server so that the robot can be controlled by more than one applications without resetting the remote application address in the interface program. The flow chart of the interface program on the interface device is shown in Figure 2.5.1.

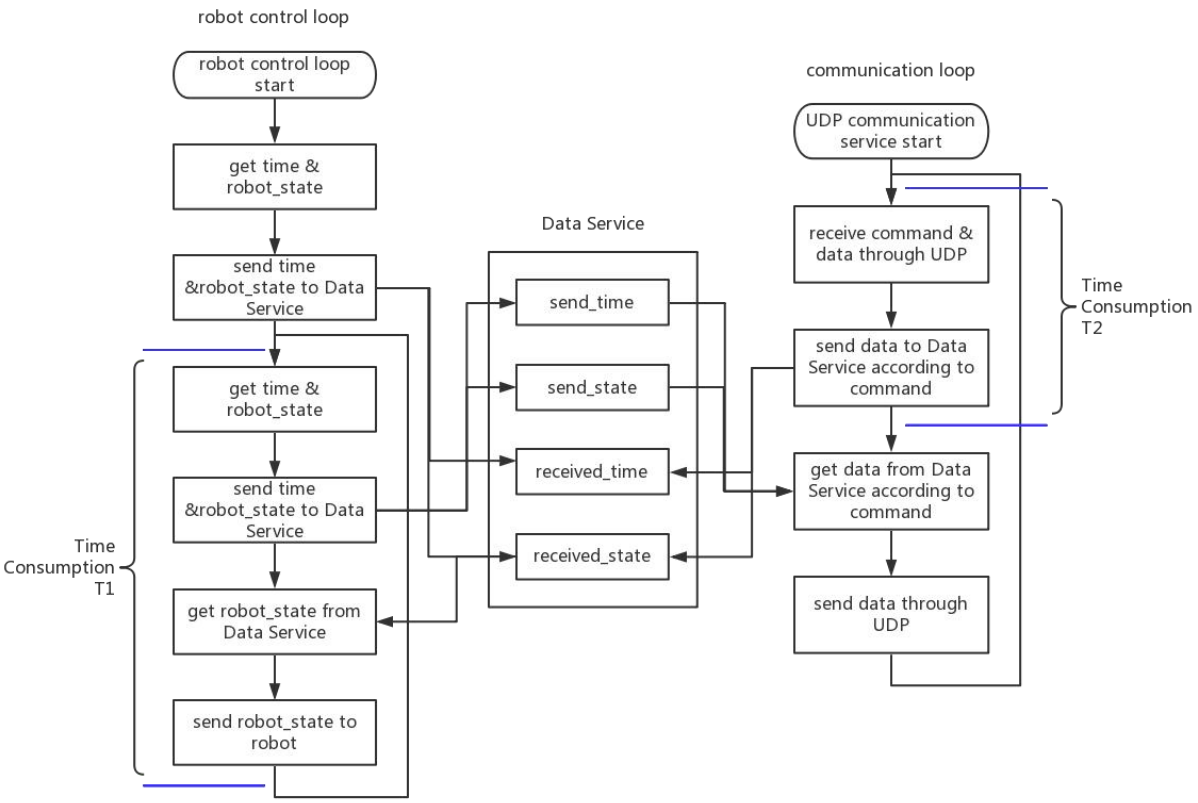


Figure 2.5.1 The Flow Chart of Interface Program

Where the UDP communication loop and the robot control loop are running separately in two threads. In the robot control loop, the program just firstly put the current time and robot_state to the “Data Service”, then get the control values from the “Data Service” and then control the robot. The thread of UDP communication can according to the received command from the application automatically change the control values in the “Data Service” or send the current time and robot_state back to the application.

The flow chart of the application program is like in Figure 2.5.2. The application program firstly send the command to get the current robot_state and time from the interface, these two values will be regarded as the initial pose and time in the following steps. Then send a another

command to get a new time from the interface. Finally, the application should calculate the control values according to the current time and the initial time and pose then send these control values back to the interface program.

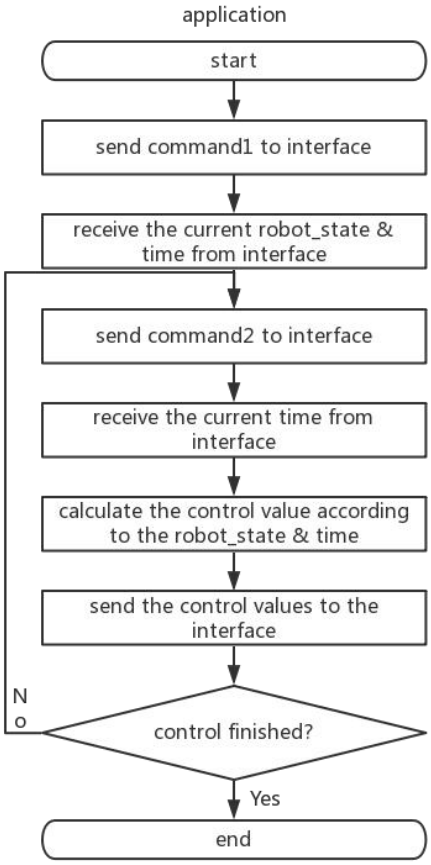


Figure 2.5.2 Flow Chart of Application Program

2.5.2. The Pros and cons

The Advantage of this UDP communication method is that it spends really very little time in robot control loop. The disadvantage is that the delay of the command is been increased by the multi-threading. The maximum delay of the received command and data is

$$T=T1+T2$$

Where the T1, T2 are the time consumption in Figure 2.5.1.

2.5.3. The Performance

With the multi-threading, the time consumption should be very small in this program. The time consumption of each part in this UDP communication method in the interface program is shown in Figure 2.5.3.

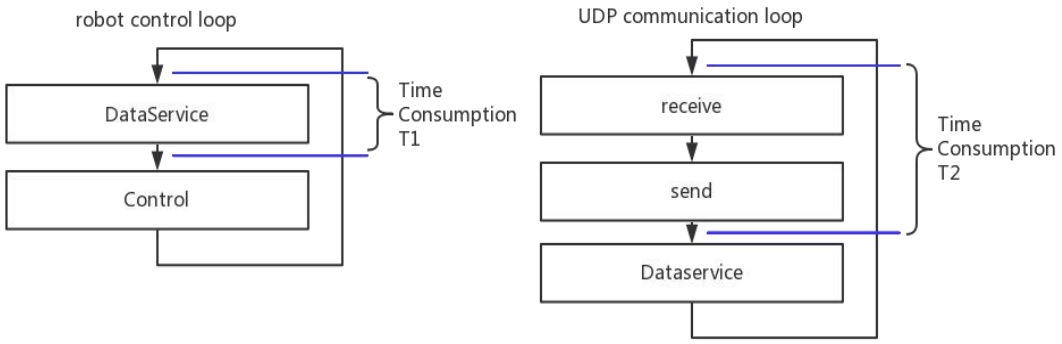


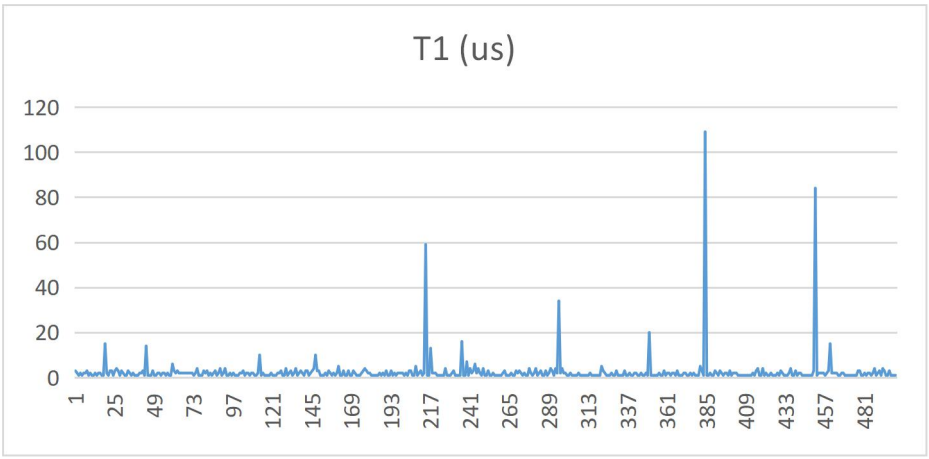
Figure 2.5.3 Time consumption in the interface program

In the Figure 2.5.4, there are the result of 500 samples of value T1, T2 in the robot control loop The maximum, minimum and the average values are shown in Figure 2.5.4.

time(us)	T1	T2
average	2.508	119.798
max	109	8854
min	1	35

Figure 2.5.4 Time Consumption of the Interface

The Figure 2.5.5 shows the time consumption T1, T2 in each samples, this figure is more intuitively, that the T1 and T2 are not stable, and time consumption in the robot control loop T1 is always less than 120us, which should be allowed in the robot control loop.



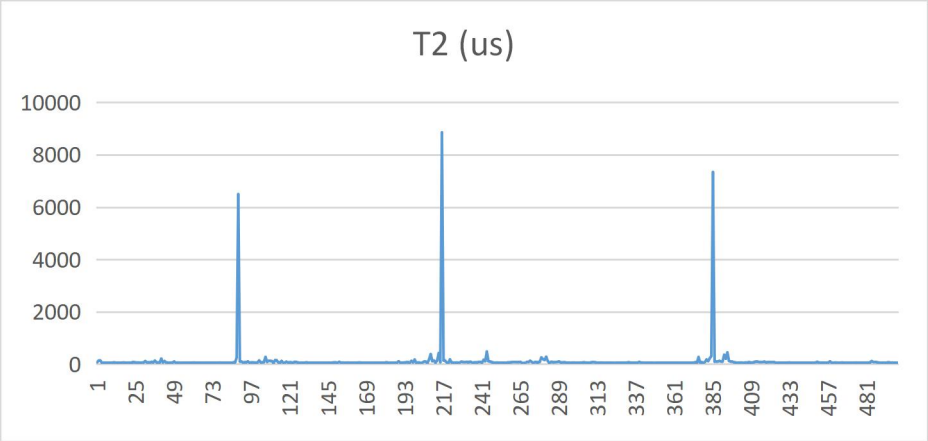


Figure 2.5.5 Time consumption T1, T2 in each samples

2.5.4. The Results

The performance seems good, but when the program tested with the robot, the robot still aborted. This time, the error showed that there were something wrong with the acceleration. The reason is possibly that I have remove the function “time=+period().toSec” which seems to be very important in the robot control loop to control the acceleration. Because when I replaced the period.toSec() of a constant 0.001 in the example “generate_cartesian_pose_motion”, the robot also aborted by the unexpected acceleration.

To solve this problem, it is suggested to achieved the UDP communication within one control loop. So in the next step, I plan to use the fastest and simplest UDP communication in the application and interface program.

2.6. The simplest and fastest synchronous UDP Communication

In this program, the simplest and fastest ways of UDP Communication are used. And at this time, the interface works as a client in order to reduce the time consumption in the robot control loop, because it doesn’t need to wait for the command from application at first.

2.6.1. The Structure of the UDP Communication

The flow chart of the simplest and fastest UDP communication is like in Figure 2.6.1. The interface application firstly send its states and time to the application and then receive the control values from the application. The information to send or received is only time and robot_state.O_T_EE_c, which contains 136 bytes. The boost library is used, and it may be the fastest way to send or receive the information through UDP.

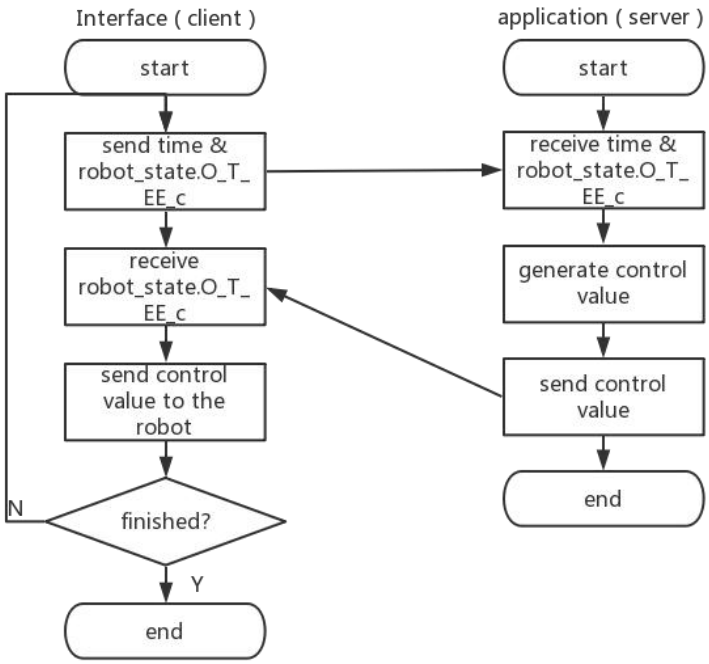


Figure 2.6.1 The Simplest and Fastest UDP Communication

2.6.2. The Pros and cons

The advantage of this method is that it may be the fastest way of UDP communication, the disadvantage is that the UDP communication seems not so stable, sometimes the sending and receiving can cost much more time than the average, which may causes problems.

2.6.3. The Performance

To test the performance of the simplest and fastest UDP communication. A timer is added in the interface program. The time consumption of each part in the robot control loop in interface program is shown in Figure 2.6.2.

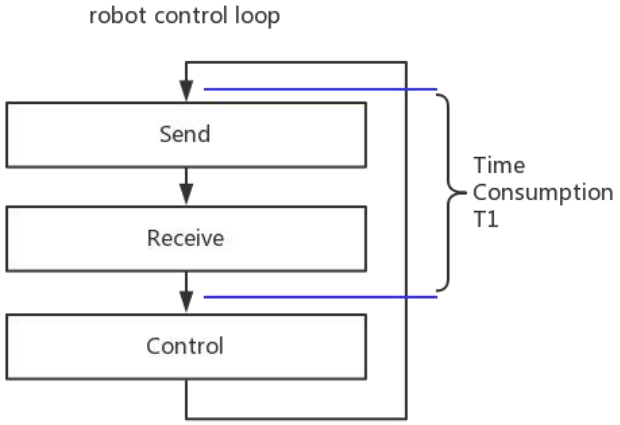


Figure 2.6.2 Time consumption in the robot control loop

After 1000 tests the result is in Figure 2.6.3.

time(us)	Using Boost	Based on C
average	106.083	83.957
max	491	421
min	63	55

Figure 2.6.3 Time consumption of the UDP communication

Where the based on C method is the UDP communication method wrote by myself and used in the programs in other sections. Then, the time consumption in each tests is in Figure 2.6.4.

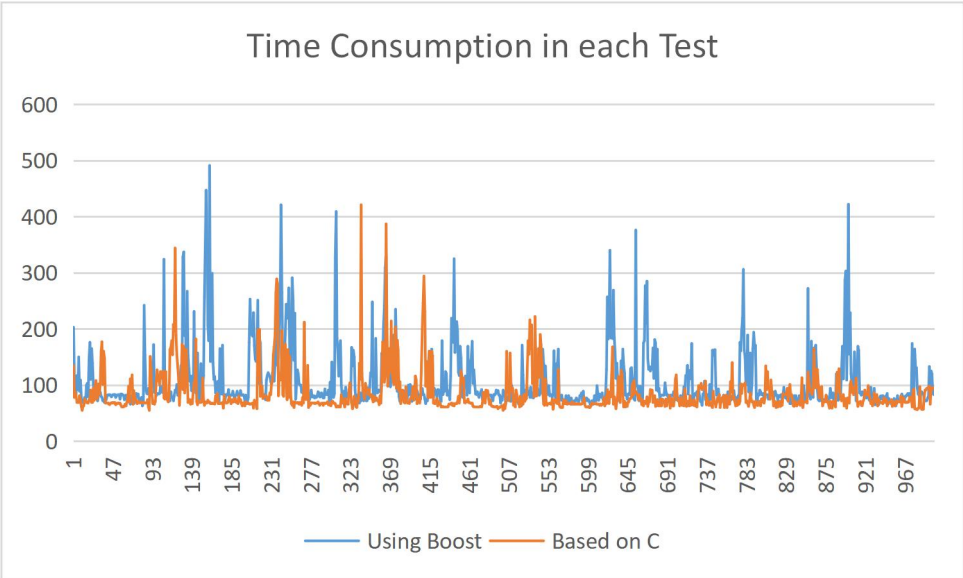


Figure 2.6.4 The Time Consumption in each Tests

It seems the boost library may a little slower than the pure C method. The average of the time consumption of these methods is wonderful, but the both UDP communications seems unstable, because the maximum value can be up to 400 us or more.

2.6.4. The Results

When this program ran with the robot, the robot still aborted by unexpected acceleration or the discontinuity. The reason may be the instability of the UDP communication and the poor hardware of the Workstation NUC.

To test the max acceptable constant delay in the robot control loop. I have tested different constant delays in each robot control loop in the example "generate_cartesian_pose_motion". Finally, I found that the acceptable constant delay is only 80us, while according to the Figure 2.6.4, the delays sometimes have exceeded 80us for a while (more than 20 cycle), which may cause the error.

2.7. The asynchronous UDP Communication - 2

To improve the stability of the UDP communication, an new synchronous method is introduced. In this method, the interface program works also as a client.

2.7.1. The Structure of the UDP Communication

The flow chart of this method is shown in Figure 2.7.1. In the interface program, there is a asynchronous UDP receiver that can automatically receive the control values from the application. When the value of sendcount is 0, that means the received control values are calculated by the information, which are sent in the same loop, so that the program can use this control value to control the robot. Otherwise, the program will give up this received value and change the time to the previous value. The application program is the same program as the above example.

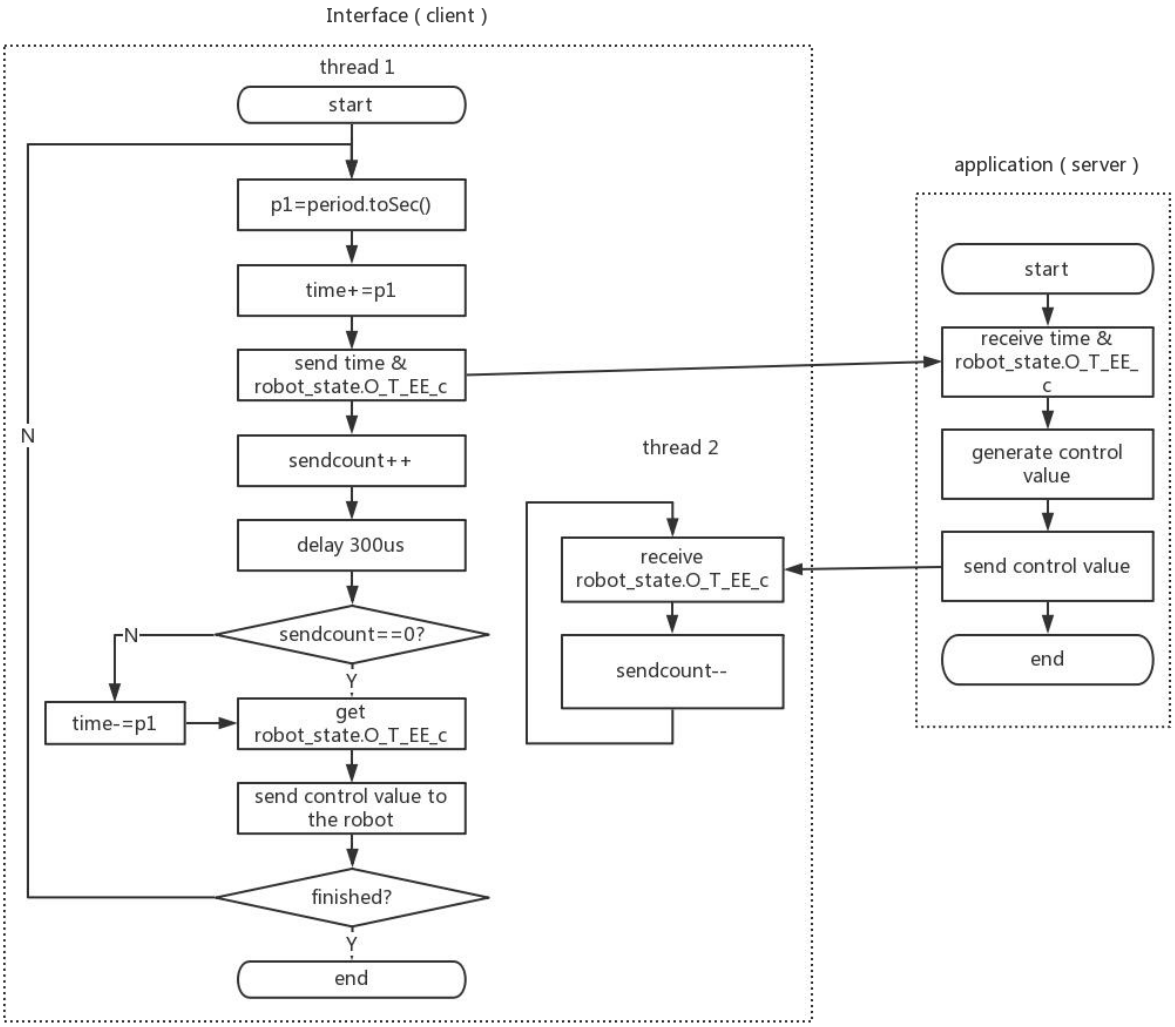


Figure 2.7.1 The asynchronous UDP Communication - 2

2.7.2. The Pros and cons

The advantage of this UDP communication is that the delay of the communication should be very stable in the robot control loop, the disadvantage is that the lost of the packets in UDP communication could be strongly increased.

2.7.3. The Performance

The approximately time consumption in the robot control loop is shown in Figure 2.7.2.

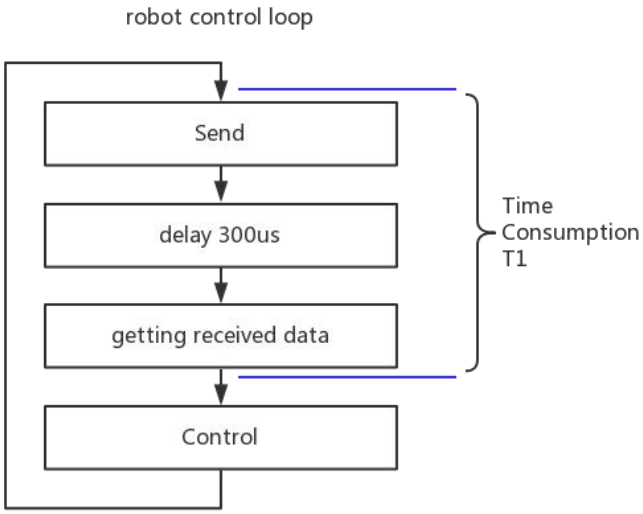


Figure 2.7.2 Time Consumption in Control Loop

The result of time consumption is shown in Figure 2.7.3. The sample size is 1000.

	time(us)
average	309.908
max	1279
min	300

Figure 2.7.3 Time Consumption of the UDP communication

The time consumption in each sample is shown in Figure 2.7.4

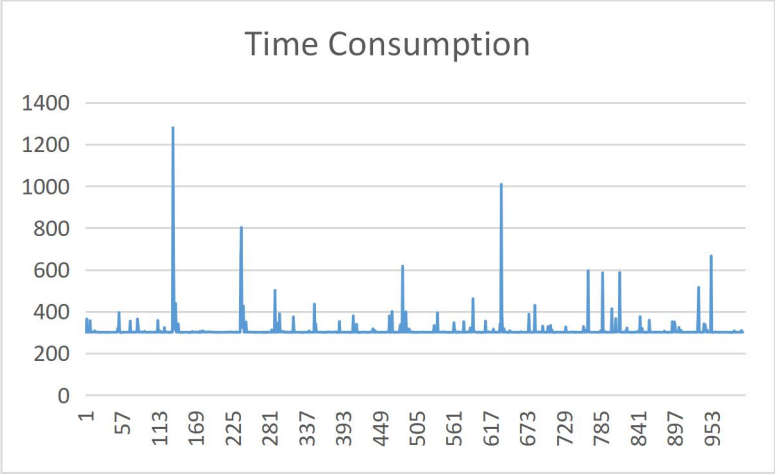


Figure 2.7.4 The Time Consumption in each Sample

Unfortunately, the performance of this method is much worse than the easiest UDP communication. It costs more time and is more unstable.

2.7.4. The Results

The results is without any doubt that the robot aborted. I think the reason is similar to the easiest and fastest UDP communication method, i.e. the delay and poor hardware of NUC.

2.8. The asynchronous UDP Communication - 3

In the libfranka, a specifically defined time, which is grown by the function period.toSec() are used in all the examples of controlling the robot. The pose of the robot are moved according to this specifically defined time. So, it is possible that the application control the robot by "time" rather than robot_state.

2.8.1. The Structure of the UDP Communication

In this method, the interface program works as a server. It receive the command time from the application. Then the current time will grows or decreases according to the command time, i.e. the application will not directly control the current pose of the robot but send to the interface a desired pose, and the speed of movement of the robot will be controlled by interface instead of the application. The code is,

```
double time_interval= command_time- current_time;
if(time_interval>period)
{
    current_time+=period;
    return 1;
}
else if(-time_interval>period)
{
    current_time-=period;
    return -1;
}
else
{
    return 0;
}
```

Where the value of period is from the function period.toSec().

The flow chart of the program is shown in Figure 2.7.1. Its structure is very similar to the

asynchronous UDP Communication - 1 in section 2.5, but this time the robot will send nothing to the application. Because of the limit time, this program has been greatly simplified and is just used for testing this method of UDP communication can be used in the interface program or not.

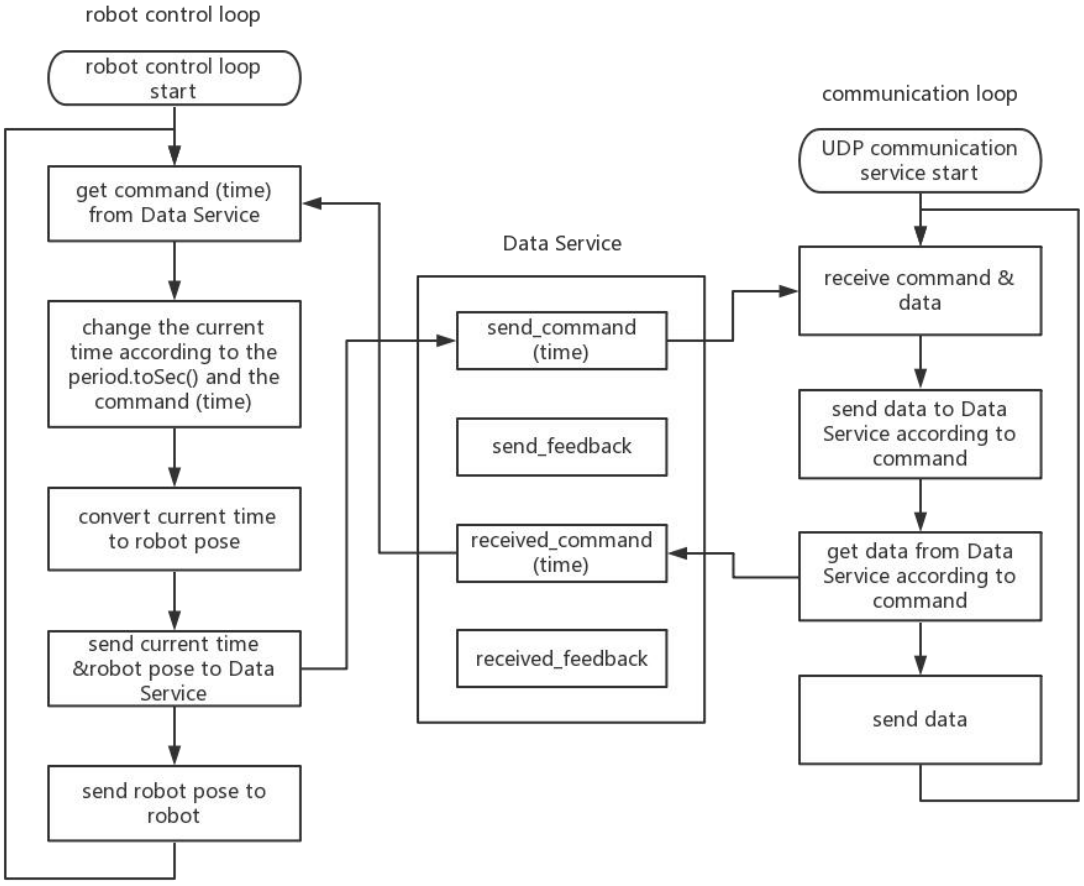


Figure 2.7.1 The asynchronous UDP Communication - 3

2.8.2. The Pros and cons

The advantage and disadvantages of this UDP communication method is similar to the first asynchronous UDP communication, i.e. the small time consumption in the robot control loop and the large latency of the command and feedback. Additionally, because of that the robot is mainly controlled by the interface program and the application only give a finally point, this program should be more stable than the programs in above.

2.8.3. The Performance

Because the structure of this program is very similar to the first asynchronous UDP communication method, the performance of them is also very close. The time consumption of each part in this UDP communication method in the interface program is shown in Figure 2.8.2.

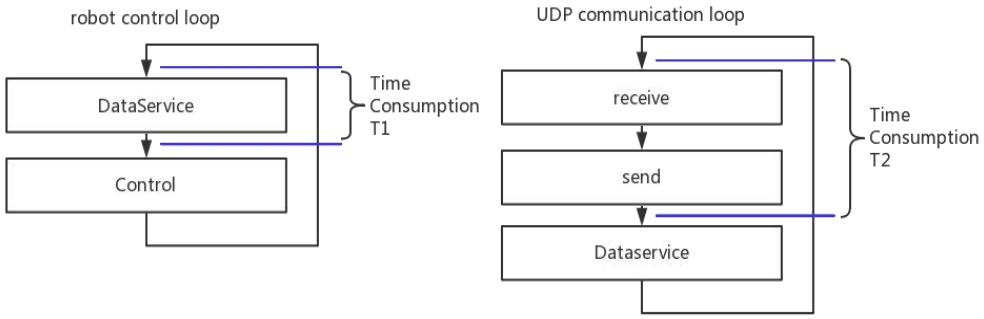


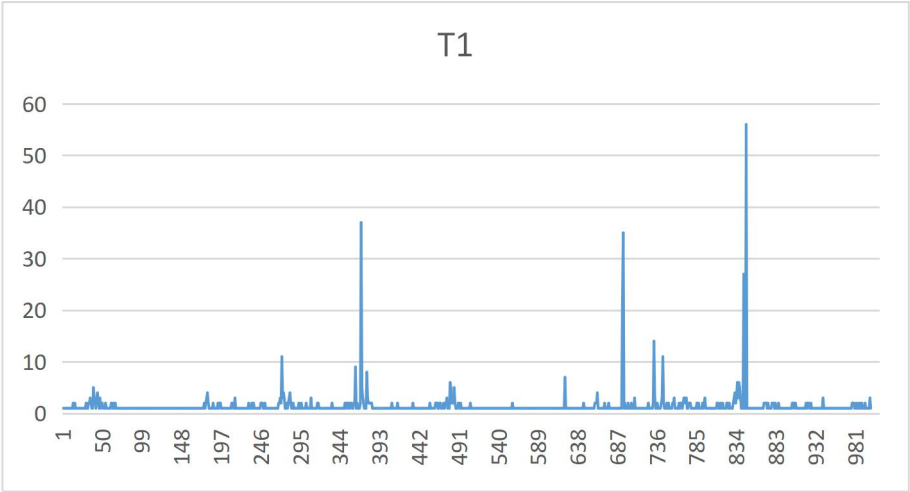
Figure 2.8.2 Time Consumption of each Part in the Interface Program

The result of time consumption is shown in Figure 2.8.3. The sample size is 1000.

	T1	T2
average	1.483	48.146
max	56	1079
min	1	30

Figure 2.8.3 The Result of the Time Consumption T1, T2

The values T1, T2 in each sample are shown in Figure 2.8.4.



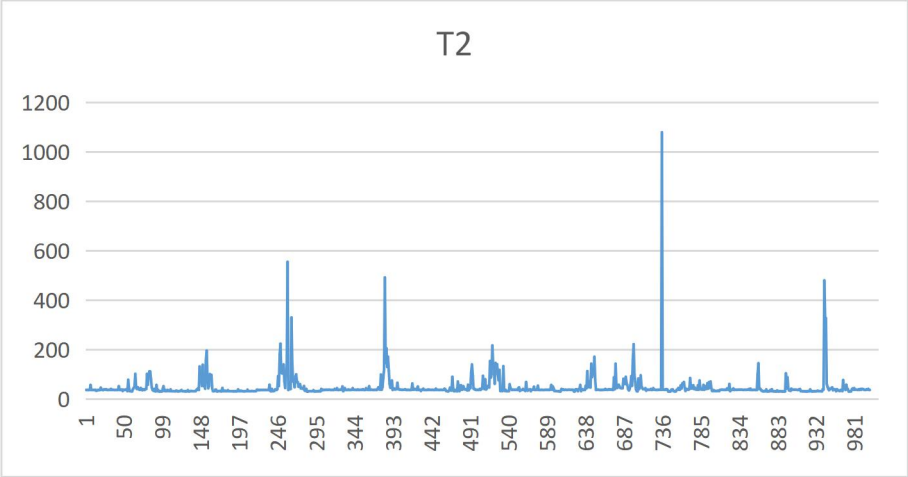


Figure 2.8.4 Time consumption T1, T2 in each samples

2.8.4. The Results

This may be the only program that can run with the robot. The delay of the communication has been successfully solved in this program. When the communication rate between the interface and the application up to 1KHz, the interface are also works well without abortion. But there is one another problem, that the program can only run successfully when the application send a large command time to the interface at first, When the value of current time of the robot grows or decreases close to the value of command time, the robot will abort by the acceleration.

3. Problem Discussion

After a bulky test programs, the time consumption has been successfully solved by the 3rd asynchronous UDP communication method. And the only left problem is the “acceleration” problem. And at this time with the help of the last program, it is realized that there is a very important part, the acceleration, was not taken very seriously in the past. Normally, as long as the speed is low enough, the influence of the acceleration should can be ignored, but the value of the acceleration and jerk are much strongly limited than expected.

In all of the above programs, when the command of the application is stopped or lost just for one control loop, the state of robot will at the previous position, so the speed of the robot will suddenly be zero, which lead to a large value of acceleration and jerk, which can exceed the limit. To clarify how this happens, a simulated program is used to be test the change of the robot's motion parameters. The code is,

```
int main()
{
    double time=0.0;
    double angle_x1=0.0;//Angle
    double angle_x2=0.0;
    double angle_dx1=0.0;//Angular velocity
    double angle_dx2=0.0;
    double angle_ddx1=0.0;//Angular acceleration
    double angle_ddx2=0.0;
    double angle_dddx1=0.0;//Angular Jerk
    for(int i=0;i<5000;i++)
    {
        angle_x1 = M_PI / 4 * (1 - std::cos(M_PI / 5.0 * time));
        if(i!=2500)time+=0.001;
        angle_dx1 = (angle_x1-angle_x2)/0.001;
        angle_x2 = angle_x1;
        angle_ddx1 = (angle_dx1-angle_dx2)/0.001;
        angle_dx2 = angle_dx1;
        angle_dddx1 = (angle_ddx1-angle_ddx2)/0.001;
        angle_ddx2 = angle_ddx1;
    }
}
```

Where the period is set to a constant 0.001s. It simulate a case that the communication only lost the command for one control loop in the interface program. The change of the robot's motion parameters is shown in following Figure 3.1-3.3.

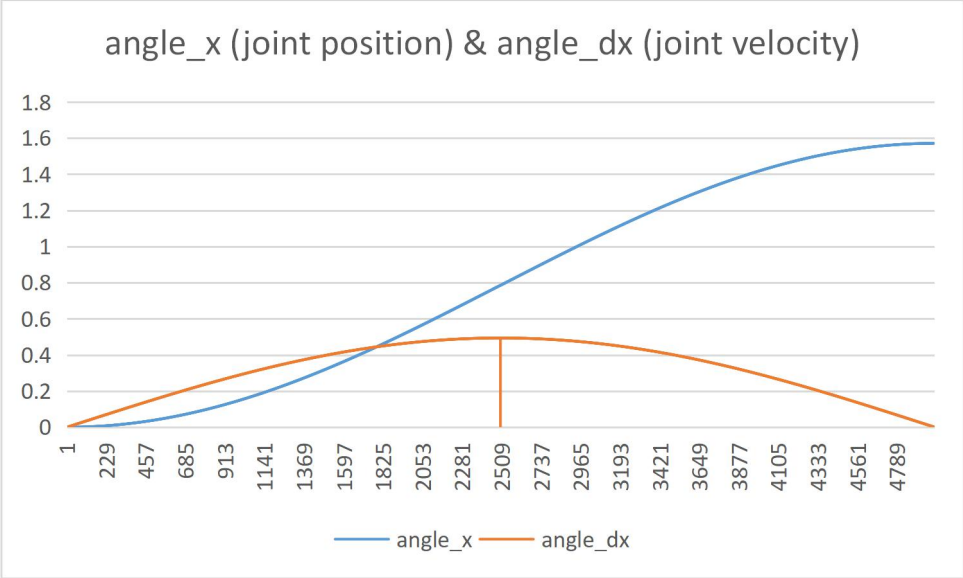


Figure 3.1 Change of Joint Position (rad) and Velocity (rad/s)

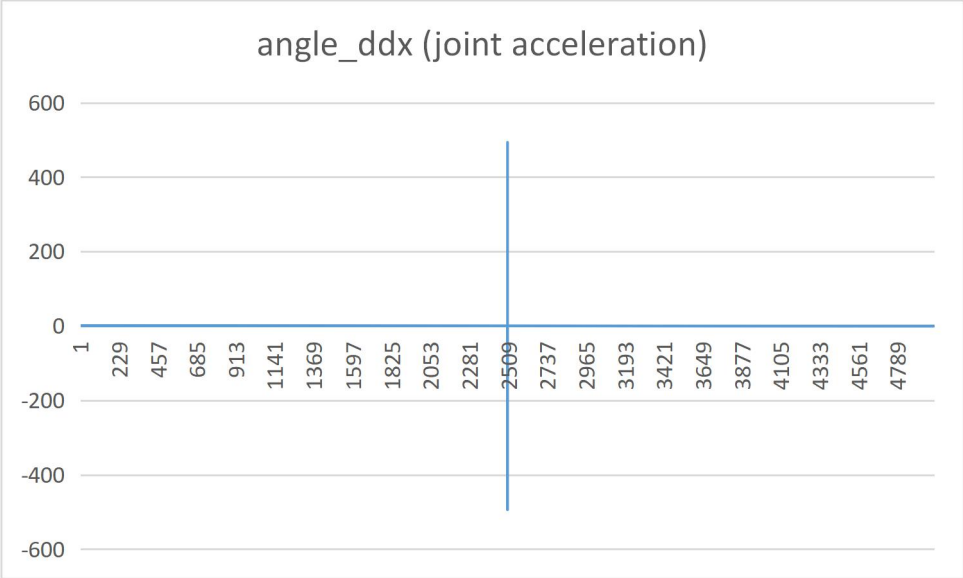


Figure 3.2 Change of Joint Acceleration (rad/s^2)

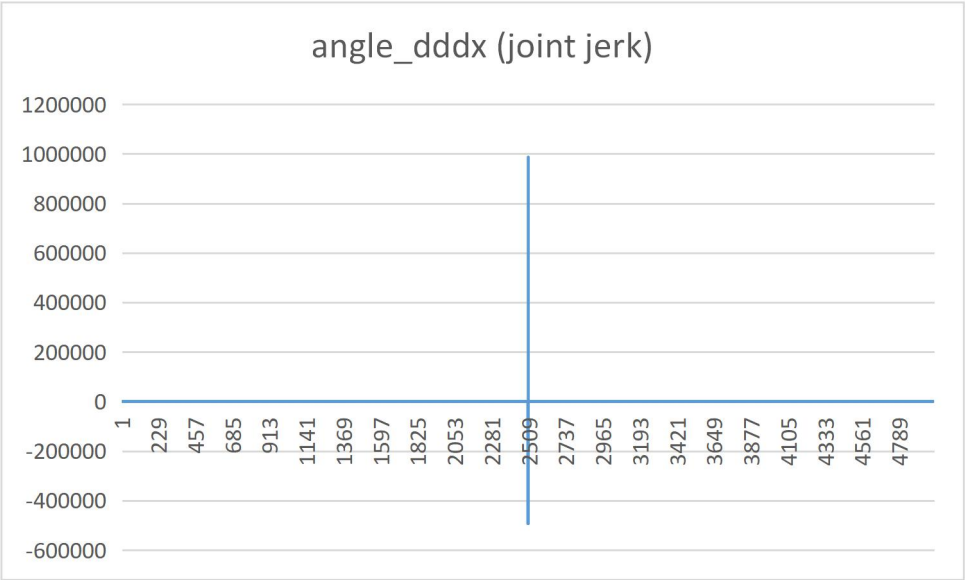


Figure 3.3 Change of Joint Jerk (rad/s^3)

Meanwhile the change of the robot's motion parameters in a ideal system, which has no delay or lost of packets, is like in Figure 3.4.

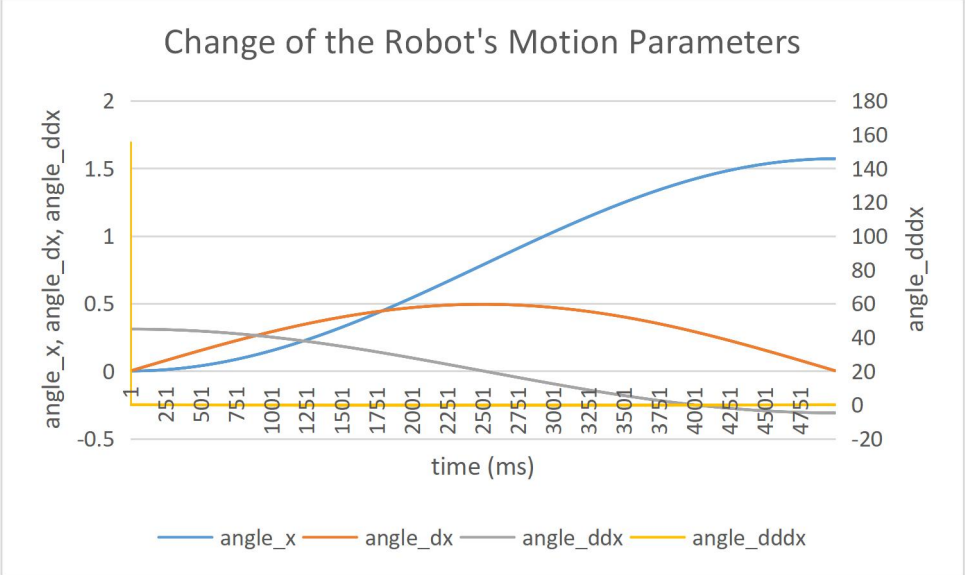


Figure 3.4 Change of the Robot's Motion Parameters in ideal Case

In addition, the limit of the robot from the official website is shown in Figure 3.5.

Name	Translation	Rotation	Elbow
$\dot{p}_{max}$	1.7000 $\frac{\mathrm{m}}{\mathrm{s}}$	2.5000 $\frac{\mathrm{rad}}{\mathrm{s}}$	2.1750 $\frac{rad}{\mathrm{s}}$
$\ddot{p}_{max}$	13.0000 $\frac{\mathrm{m}}{\mathrm{s}^2}$	25.0000 $\frac{\mathrm{rad}}{\mathrm{s}^2}$	10.0000 $\frac{rad}{\mathrm{s}^2}$
$\dddot{p}_{max}$	6500.0000 $\frac{\mathrm{m}}{\mathrm{s}^3}$	12500.0000 $\frac{\mathrm{rad}}{\mathrm{s}^3}$	5000.0000 $\frac{rad}{\mathrm{s}^3}$

Figure 3.5 Limits in the Cartesian space [10]

Where the simulated values in the first case are much more bigger than the limit, while the values in the ideal situation are completely under the limits. Because the acceleration and the jerk are strongly limited, it is very difficult to control the robot directly by a determined pose received from the application through UDP, which will lost packets in the communication and lead to large acceleration and jerk. So the new pose should be calculated by current position, speed and acceleration, and it is important to makes the robot mildly speed up and slow down.

4. Conclusions

This project has introduced two ways to reduce the traffic in the control loop, one is reduce the quantity of data through the UDP communication, another is improve the UDP communication to reduce the time consumption of the communication in the control loop. Firstly, a data extraction method and a special RLE Compression method is introduced to reduce the quantity of data. Secondly, several UDP communication method are introduced to reduce the traffic in the control loop with the robot, the performances of some methods are not bad, but there are still a lot of problems with the control of robot. Because of the limited time, I can't solve all of them. And from these failures, you can get the following tips.

1. The failure of the first synchronous UDP Communication and the simplest and fastest synchronous UDP Communication shows that a synchronous UDP Communication because of its unstable and delay could not be directly used in the robot control loop.

2. The failure of the asynchronous UDP Communication - 1 and the asynchronous UDP Communication - 2 shows that the robot can't directly controlled by a determined pose, by which, the lost of the packets could cause large acceleration and jerk exceed the limit.

3. It is important to make the robot mildly speed up and slow down, because the limit of the robot is very strict.

Finally, in this project, I have learned much about C++, UDP communication, multi-threading, RCR32 and Compression. And thanks very much for M.Sc. Elena Urbano Pérez and Dr.-Ing. Ievgenii Tsokalo, they gave me really a lot of help.

References

[1] Overview - Franka Control Interface (FCI) Documentation, Franka Emika GmbH, Retrieved 24 February 2019, from : https://frankaemika.github.io/docs/overview.html.

[2] Minimum system and network requirements - Franka Control Interface (FCI) Documentation, Franka Emika GmbH, Retrieved 24 February 2019, from : https://frankaemika.github.io/docs/requirements.html.

[3] franka::RobotState Struct Reference, Franka Emika GmbH, Retrieved 24 February 2019, from : https://frankaemika.github.io/libfranka/structfranka_1_1RobotState.html#a3e5b4b7687856e92d826044be7d15733

[4] User Datagram Protocol (10 February 2019), Wikipedia, Retrieved 24 February 2019, from : https://en.wikipedia.org/wiki/User_Datagram_Protocol

[5] Run-length encoding (3 February 2019), Wikipedia, Retrieved 24 February 2019, from : https://en.wikipedia.org/wiki/Run-length_encoding

[6] Double-precision floating-point format (23 February 2019), Wikipedia, Retrieved 24 February 2019, from : https://en.wikipedia.org/wiki/Double-precision_floating-point_format

[7] QuickLZ 1.5.x, Retrieved 24 February 2019, from : http://www.quicklz.com/

[8] Snappy - A fast compressor/decompressor, Google, Retrieved 24 February 2019, from : https://google.github.io/snappy/

[9] Zlib - A Massively Spiffy Yet Delicately Unobtrusive Compression Library (15 December 2017), Greg Roelofs, Jean-loup Gailly and Mark Adler, Retrieved 24 February 2019, from : https://www.zlib.net/

[10] Robot and interface specifications, Franka Emika GmbH, Retrieved February 24, 2019, from : https://frankaemika.github.io/docs/control_parameters.html