Yunbin Shen 4724075
Deutsche Telekom Chair for Communication Networks
TU Dresden

Reducing the Traffic in the Control Loop for the Robot Arm (OS+PBL)

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

Dresden // 04.03.2019

Contents
Reducing the Traffic in the Control Loop for the Robot Arm

1 Introduction
1.1 The Reason to reduce the Traffic

1.1.1 A lot of redundant information the class franka::RobotState [1]

— All the members in the class franka::RobotState, the size is more than 2000 Bytes.

O_T_EE	F_x_Cload	tau_J	cartesian_contact	O_ddP_EE_c
O_T_EE_d	m_total	tau_J_d	joint_collision	theta
F_T_EE	I_total	dtau_J	cartesian_collision	dtheta
EE_T_K	F_x_Ctotal	q	tau_ext_hat_filtered	current_errors
m_ee	elbow	**q_d**	O_F_ext_hat_K	last_motion_errors
I_ee	elbow_d	dq	K_F_ext_hat_K	control_command_success_rate
F_x_Cee	elbow_c	dq_d	O_dP_EE_d	robot_mode
m_load	delbow_c	ddq_d	**O_T_EE_c**	time
I_load	ddelbow_c	joint_contact	O_dP_EE_c	

1 Introduction
1.1 The Reason to reduce the Traffic

1.1.1 A lot of redundant information the class franka::RobotState [1]
— The libfranka provide ways to control the robot, in the offical examples there are only 2 members are used.

Joint Positions - robot_state.O_T_EE_c (desired joint position)

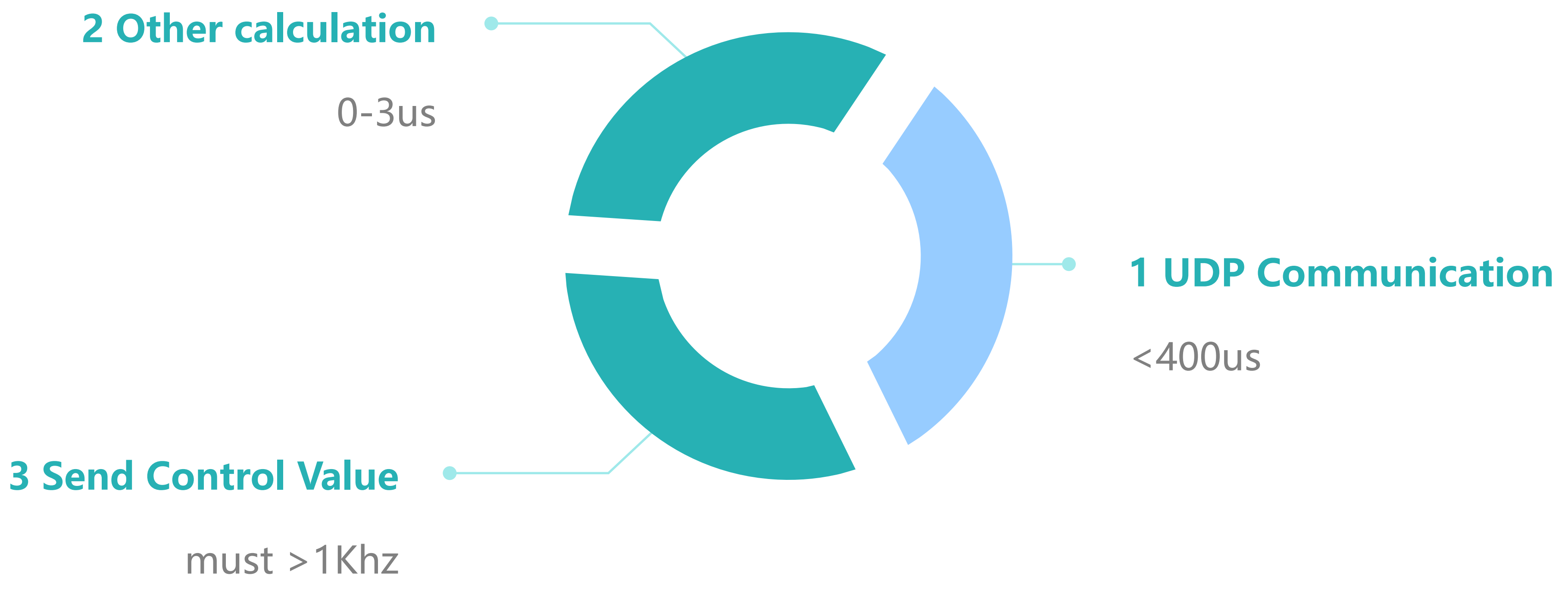
Joint Velocity

Cartesian Pose - robot_state.q_d (cartesian position)

Cartesian Velocities

1 Introduction
1.1 The Reason to reduce the Traffic

1.1.2 The limit of time consumption of UDP communication in control loop



1 Introduction
1.2 Two ways of Reducing the traffic

1.2.1 Extracting the useful data to reduce the sending/receiving data

More than 2000 Bytes

136 Bytes

1 Introduction
1.2 Two ways of Reducing the traffic

1.2.2 Improve the perfromance of UDP communication

A Using boost library [2]

B Based on C UDP [3]
Low level programing

C Asynchronous UDP communication

2 Data Processing
2.1 Data Extraction

2.1.1 Extracting the useful data

— Convert the useful data to a chary array

```
uchar8t *mStrCpy(uchar8t *&pre, uint32t copy_length, const uchar8t *dst)
{
    for (uint32t i = 0; i < copy_length; i++)
        *pre++ = *dst++;
    *(pre) = '\0';
    return pre;
}

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

2 Data Processing
2.1 Data Extraction

2.1.1 Extracting the useful data

— Convert the useful data to a chary array

```
uint32t length;
uchar8t *send_data;

DPInit();
SPAddData(robot_state.O_T_EE_c);
SPAddData(robot_state.q_d);
......
GetSendData(send_data, length);
```

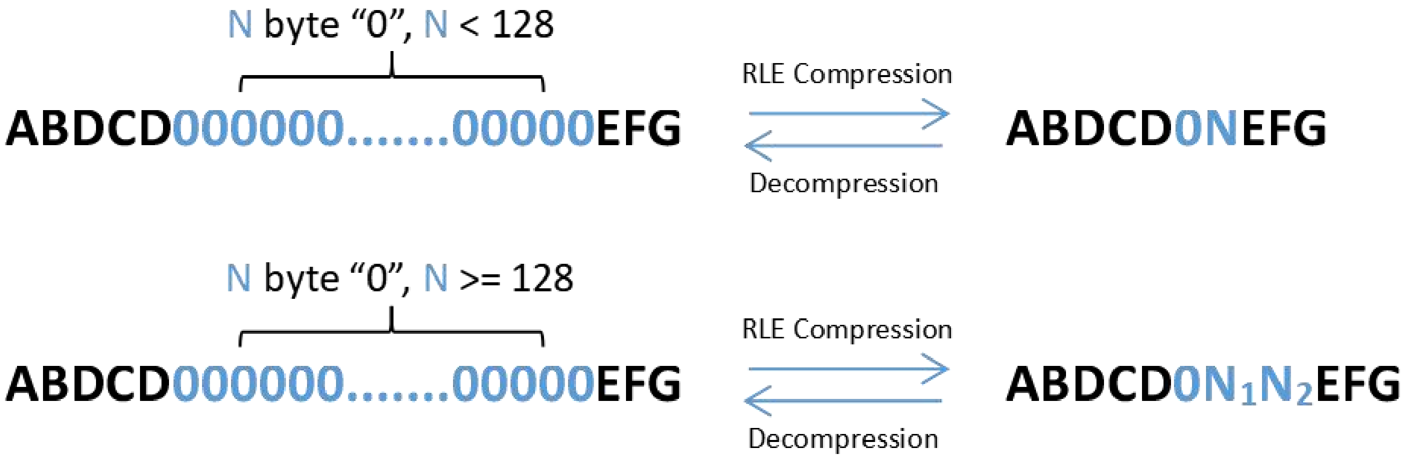
The **send_data**, **length** can be directly used by UDP communication.

2 Data Processing
2.2 Fast RLE Compression

2.2.1 The fast run-length Encoding Compression

- Further reduce the amount of data sent through the UDP

- N_1 = (N % 128) | 0x80
- N_2 = N / 128
- N is the number of zeros.



2 Data Processing
2.2 Fast RLE Compression

2.2.2 The performace of fast run-length Encoding Compression

— Tested data (std::array<double, 16>):

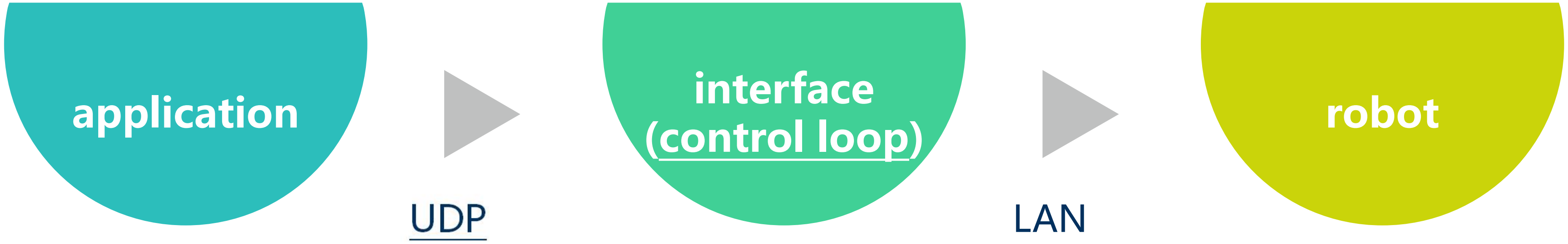
robot_state.O_T_EE_c = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -0.999023, 0, -0.00810967, 0, 0};

Compress method		Quicklz[4]	Zlib[5]	Snappy[6]	RLE
Size (bytes)	Original	128			
	Compressed	41	33	41	**24**
Time (us)		101	203	26	**1**

3 UDP Communication
3.1 The UDP Communication in the Whole System

3.1 How to improve the performance of UDP communication

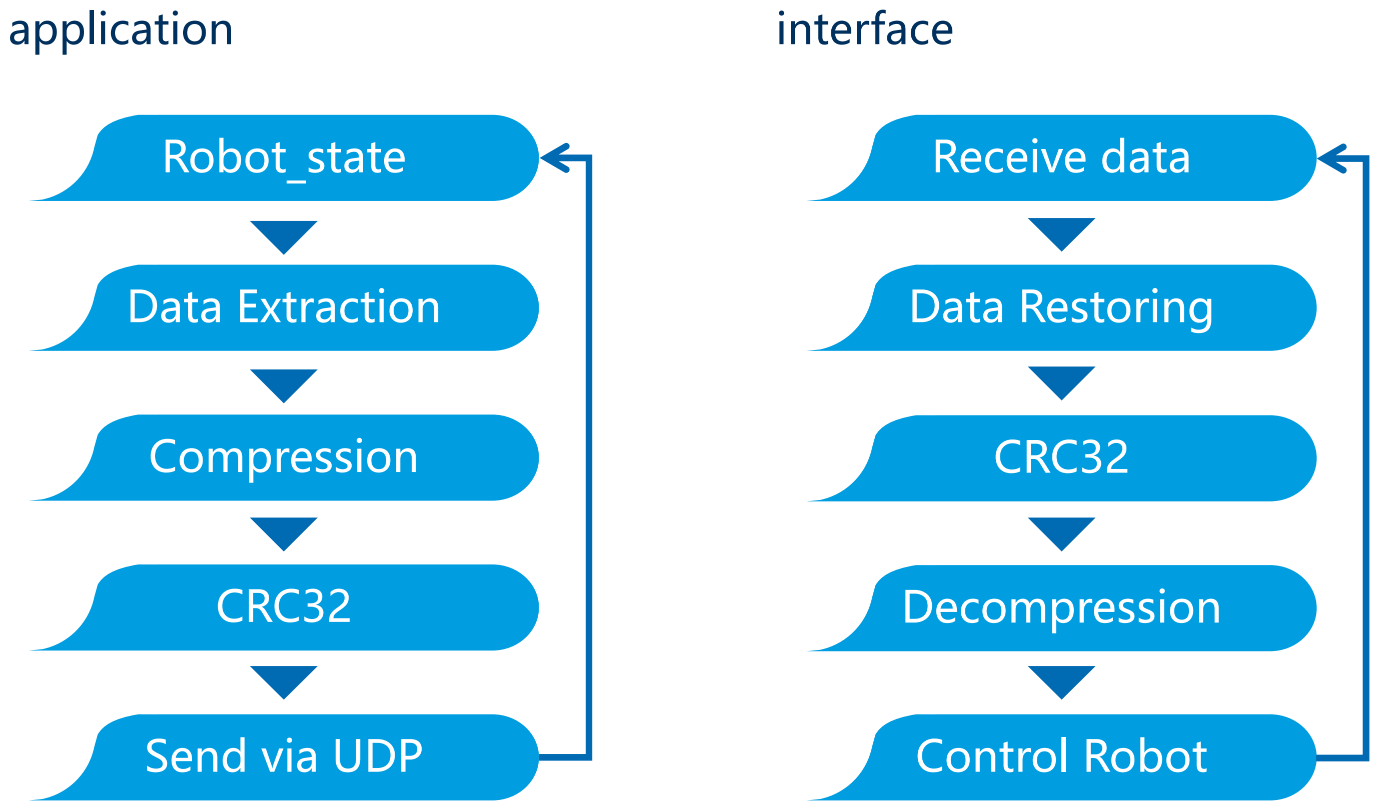
- 1. UDP communication between Application and Interface
- 2. Time consumption in the control loop



3 UDP Communication

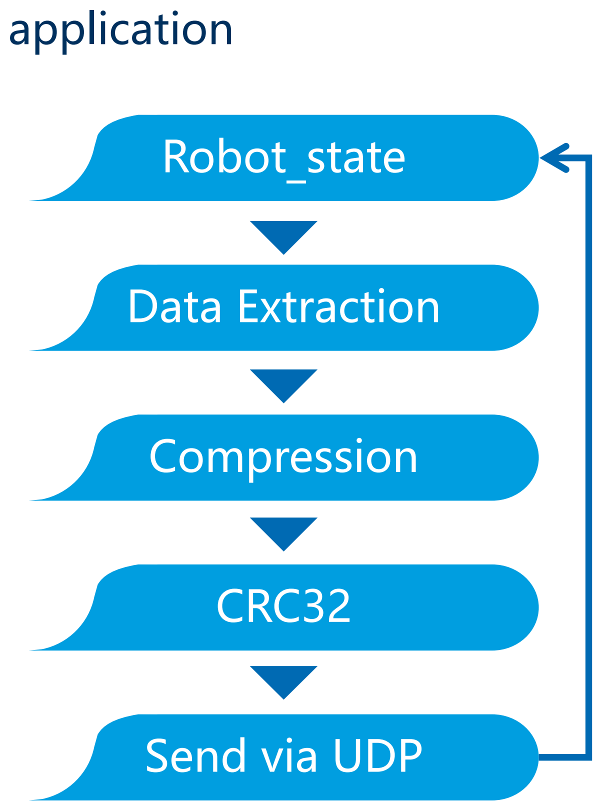
3.2 The first synchronous UDP Communication

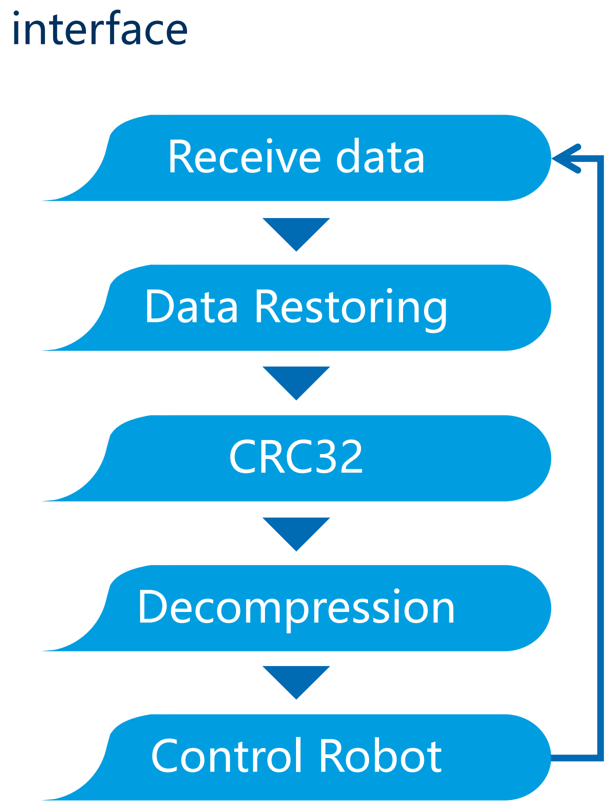
3.2.1 Structure of the first synchronous UDP commnuication



3 UDP Communication
3.2 The first synchronous UDP Communication

3.2.1 Structure of the first synchronous UDP commnuication





Advantage:

small quantity of the sent data

data reliability

Easy to set up Communication

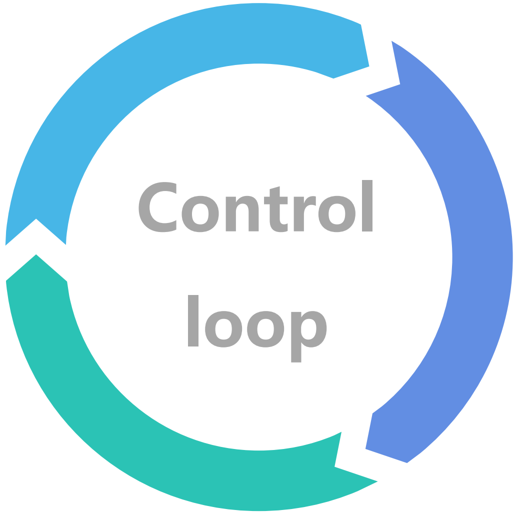
Disadvatage:

Computational complexity

3 UDP Communication
3.2 The first synchronous UDP Communication

3.2.2 Time consumption in the control loop for the robot am

1. Receive the data (400-1000us)



2. Send data (32-200us)

3. Control the Robot

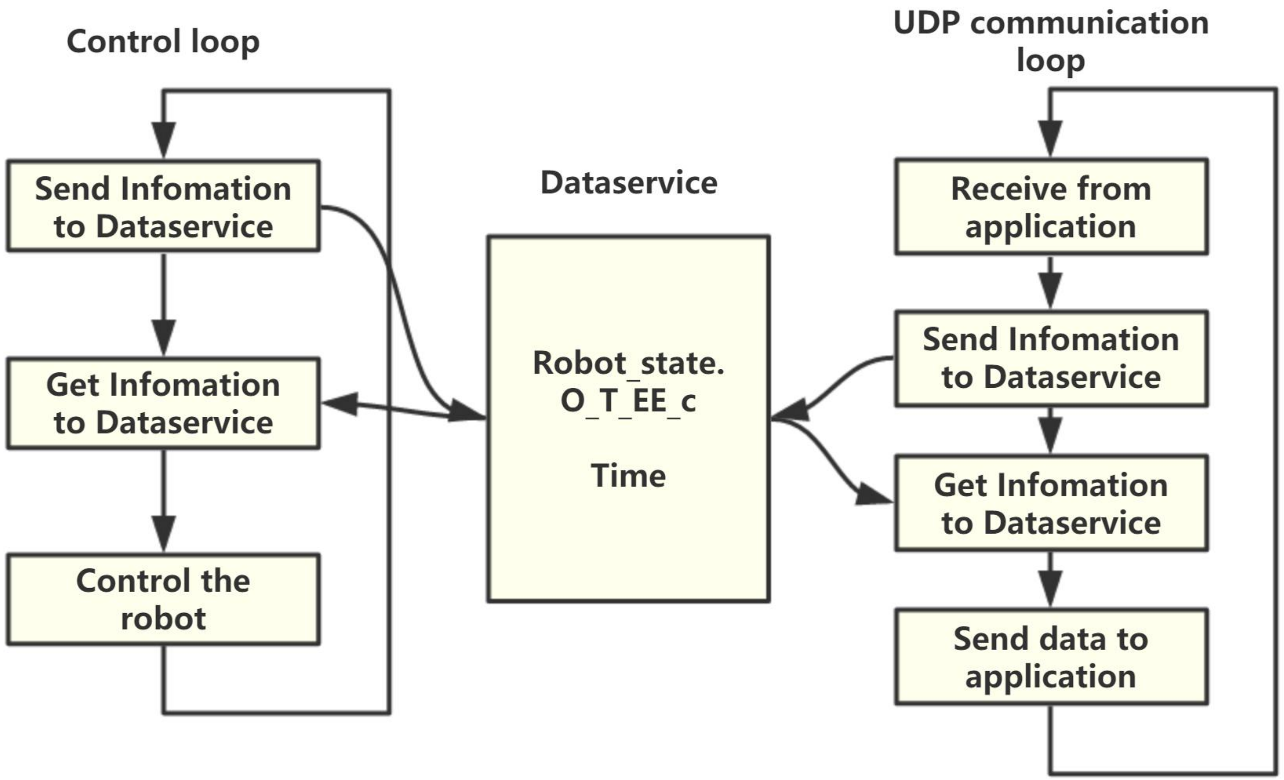
3 UDP Communication
3.2 The first synchronous UDP Communication

3.2.3 The results

- The robot arm aborted
- Reason: The receiving and sending spend too much time in the control loop.
- **Next Step:** Using asynchronous UDP communication

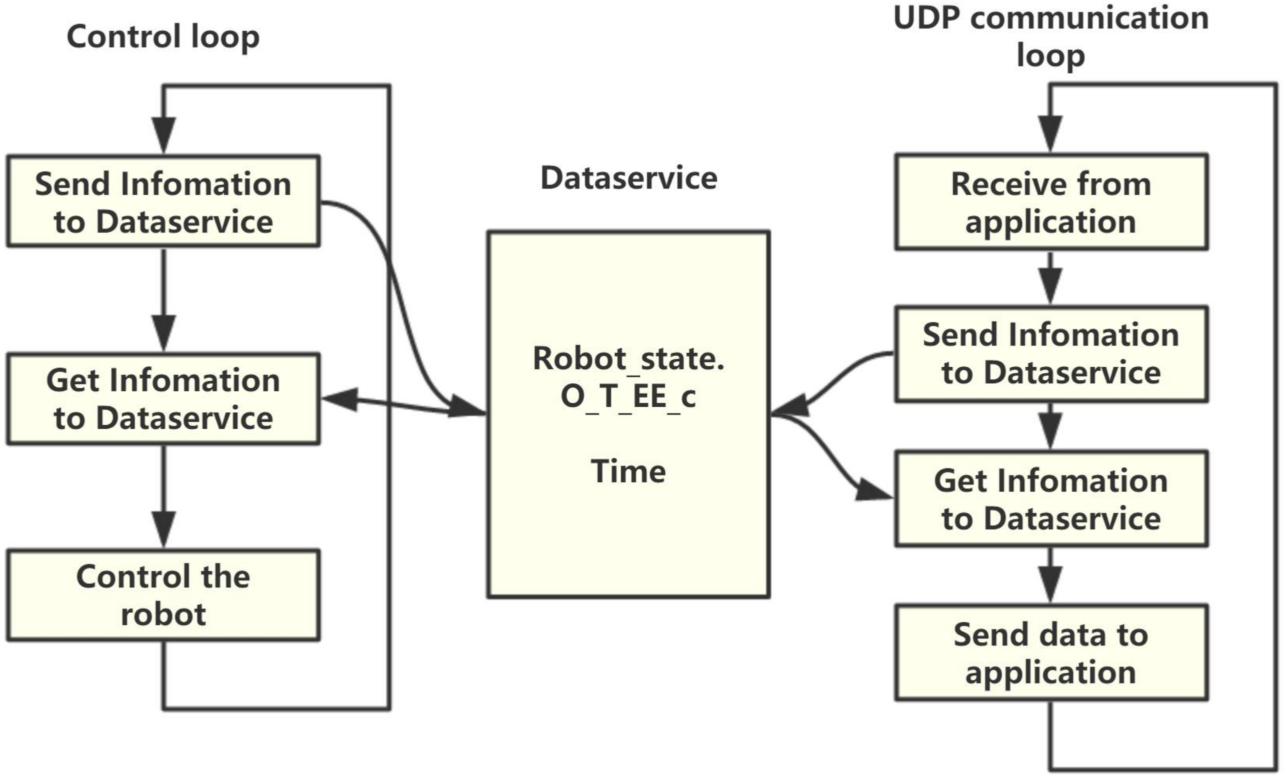
3 UDP Communication
3.3 The asynchronous UDP Communication - 1

3.3.1 Structure of the asynchronous UDP commnuication - 1
— The structure of the interface program



3 UDP Communication
3.3 The asynchronous UDP Communication - 1

3.3.1 Structure of the asynchronous UDP commnuication - 1
— The structure of the interface program



Advantage:

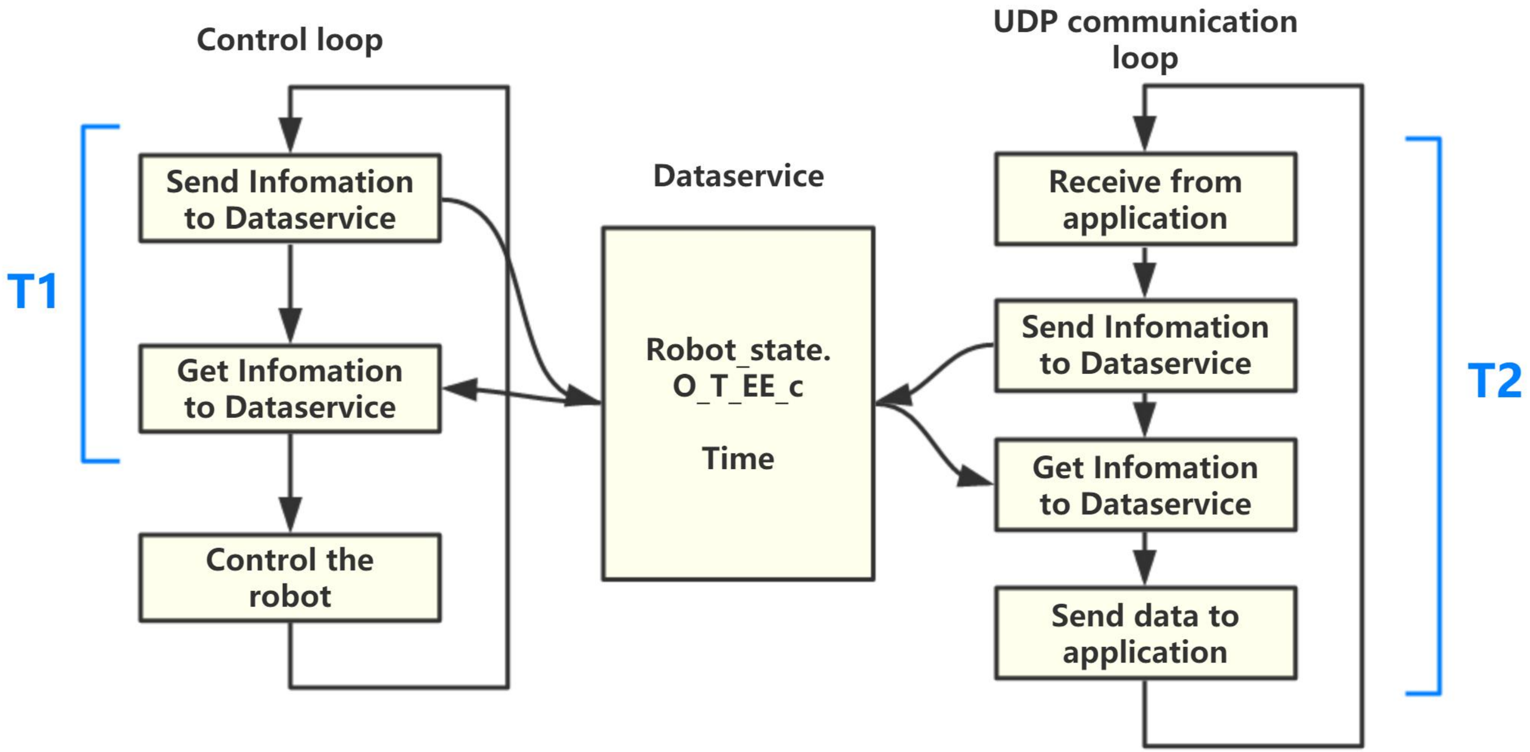
small latency in the control loop

Disadvatage:

large latency of the data through the UDP

3 UDP Communication
3.3 The asynchronous UDP Communication - 1

3.3.2 Performance of the asynchronous UDP commnuication - 1
— Test the values of T1, T2



3 UDP Communication
3.3 The asynchronous UDP Communication - 1

3.3.2 Performance of the asynchronous UDP commnuication - 1

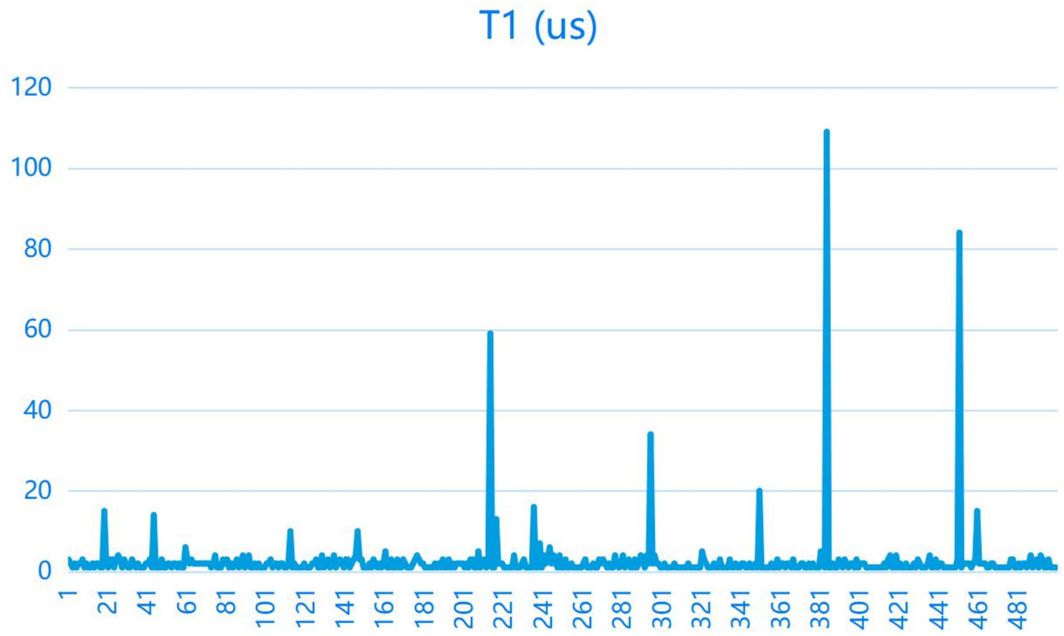
— T1, the time consumption in control loop
— T2, the time consumption communcation loop

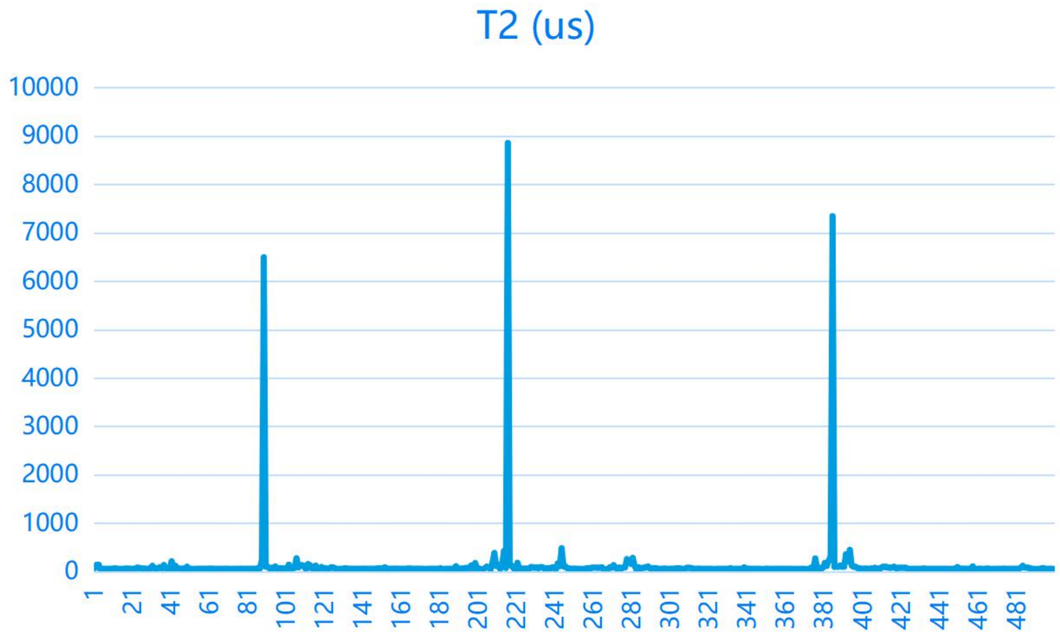
— The sample size is 500.

Time (us)	T1	T2
Average	2.508	119.798
Max	109	8854
Min	1	35

3 UDP Communication
3.3 The asynchronous UDP Communication - 1

3.3.2 Performance of the asynchronous UDP commnuication - 1
— T1, the time consumption in control loop
— T2, the time consumption communcation loop
— The sample size is 500.





3 UDP Communication
3.3 The asynchronous UDP Communication - 1

3.3.3 The results

— The robot arm aborted

— Reason: The discontinuity of acceleration, may be caused by the delay of the command

3 UDP Communication
3.3 The asynchronous UDP Communication - 1

3.3.3 The results

— The code of control loop in the **example**

```
robot.control([&time, &DS](const franka::RobotState &robot_state,
                           franka::Duration period) -> franka::CartesianPose {
    time += period.toSec();
    if (time == 0.0)
    {
        initial_pose = robot_state.O_T_EE_c;
    }
    new_pose = function(time , initial_pose)
    return  new_pose;
});
```

3 UDP Communication
3.3 The asynchronous UDP Communication - 1

3.3.3 The results

— The code of control loop in the **interface application**

```
robot.control([&time, &DS](const franka::RobotState &robot_state,
                           franka::Duration period) -> franka::CartesianPose {
    time += period.toSec();
    if (time == 0.0)
    {
        Dataservice.Init( robot_state.O_T_EE_c, time );
        Dataservice.run();
    }
    Dataservice.UpdateSendData( robot_state.O_T_EE_c, time );
    return  Dataservice.RecvState();
     // The received pose is calculated by the  "time"  updated in the previous control loop
});
```

3 UDP Communication
3.3 The asynchronous UDP Communication - 1

3.3.3 The results

- The robot arm aborted by the discontinuity of acceleration

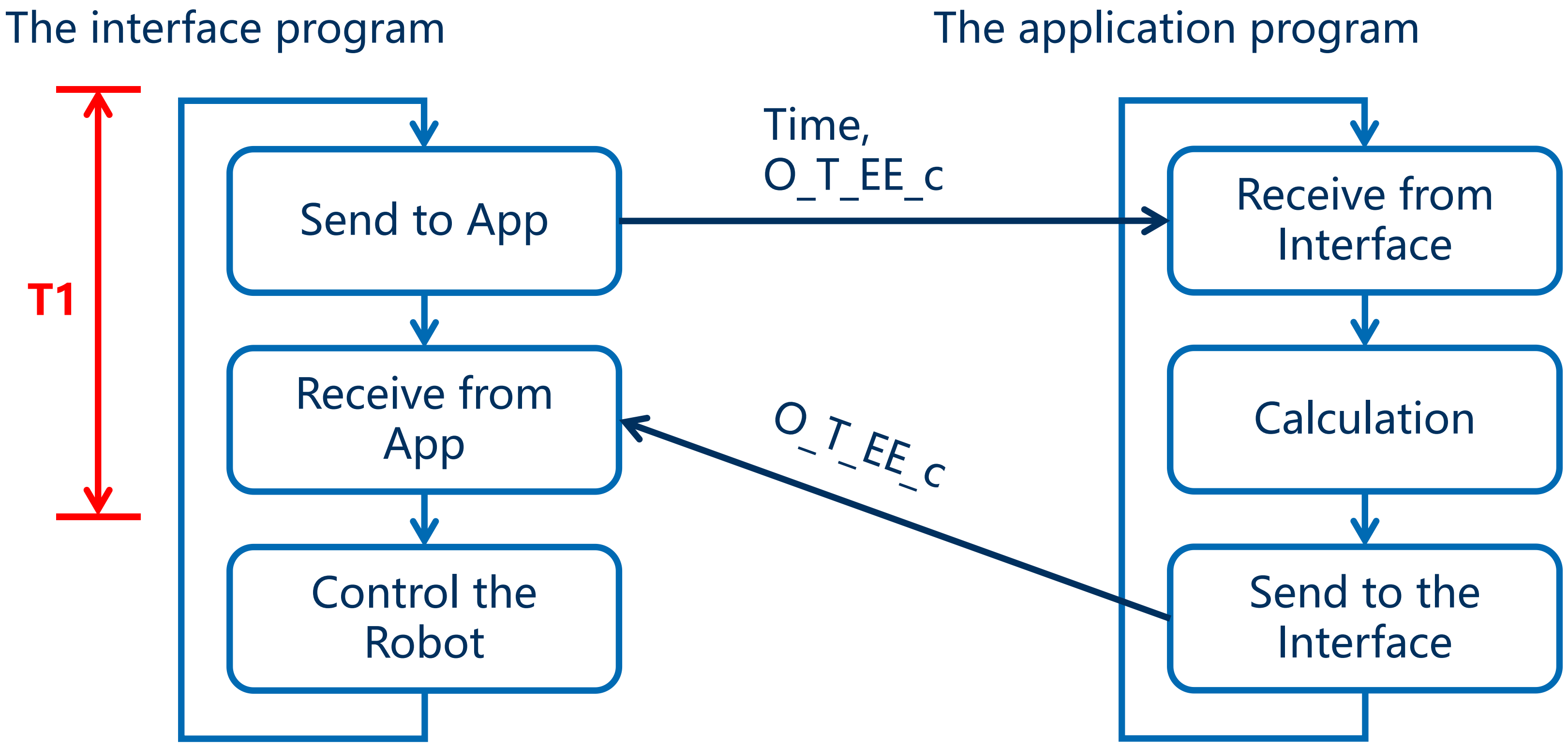
- Reason:
- 1. delay of the command
- 2. period.toSec() is not a constant

- **Next Step:** The simplest and fastest synchronous UDP Commmunication, finisched in one control loop

3 UDP Communication

3.4 The simplest and fastest synchronous UDP Commmunication

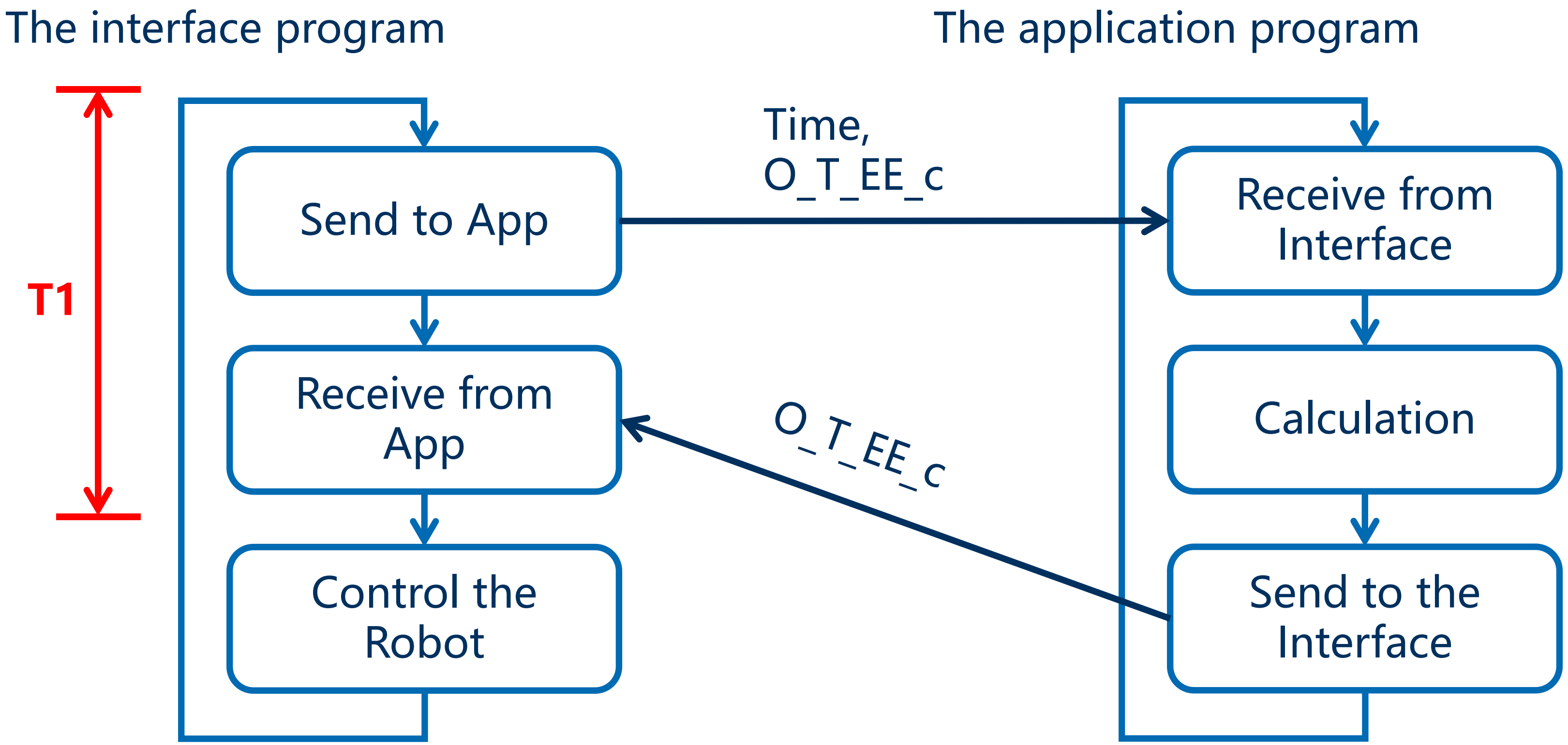
3.4.1 Structure of the simplest and fastest UDP commnuication



3 UDP Communication

3.4 The simplest and fastest synchronous UDP Commmunication

3.4.1 Structure of the simplest and fastest UDP commnuication



Advantage:

fastest way of UDP communication

Disadvatage:

Unstable

3 UDP Communication
3.4 The simplest and fastest synchronous UDP Commmunication

3.4.2 Performance of the simplest and fastest UDP commnuication

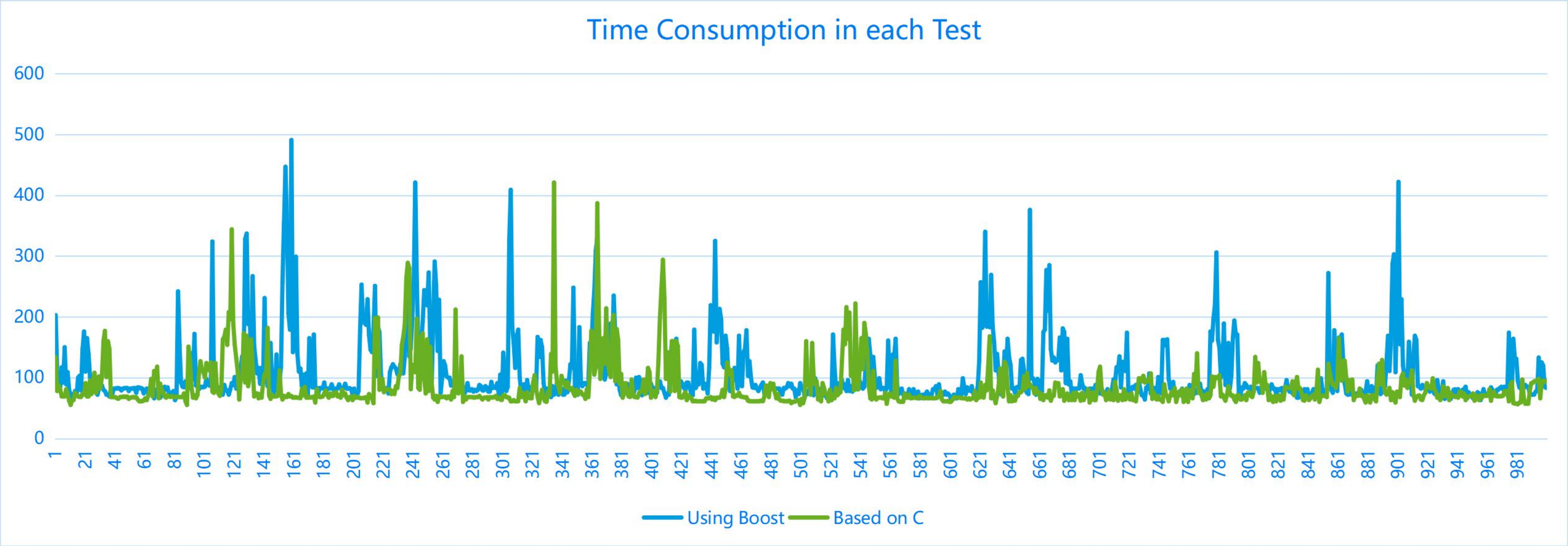
— T1, the time consumption in control loop
— The sample size is 1000.

— 2 Methods of UDP communication: boost library, C

Time T1 (us)	Using boost	Based on C
Average	106.083	83.957
Max	491	421
Min	63	55

3 UDP Communication
3.4 The simplest and fastest synchronous UDP Commmunication

3.4.2 Performance of the simplest and fastest UDP commnuication



3 UDP Communication
3.4 The asynchronous UDP Communication - 1

3.4.3 The results

- The robot arm aborted
- Reason: The discontinuity of acceleration und velocity.
- **Next Step:** To find out the maximum acceptable constant delay in the original example

3 UDP Communication
3.4 The asynchronous UDP Communication - 1

3.4.3 The results

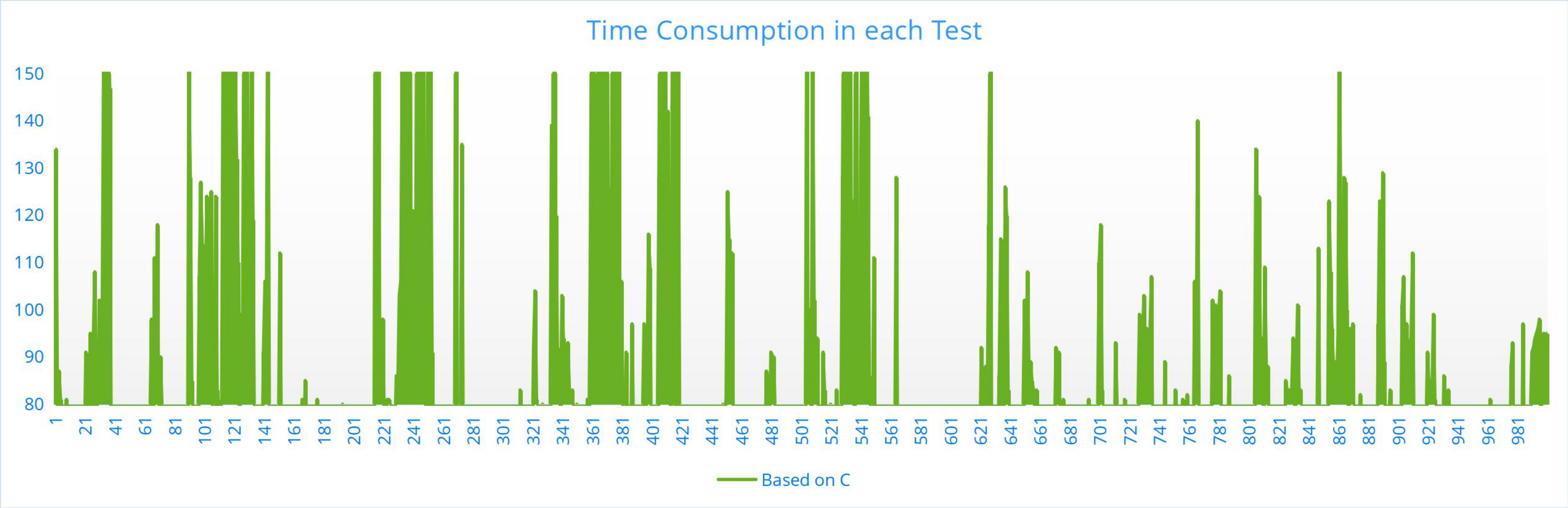
— The code of control loop in the **example**

```
robot.control([&time, &DS](const franka::RobotState &robot_state,
                          franka::Duration period) -> franka::CartesianPose {
    time += period.toSec();
    delayus(80); //The maxium acceptable value is only 80us !!!
    if (time == 0.0)
    {
        initial_pose = robot_state.O_T_EE_c;
    }
    new_pose = function(time , initial_pose)
    return  new_pose;
});
```

3 UDP Communication
3.4 The asynchronous UDP Communication - 1

3.4.3 The results

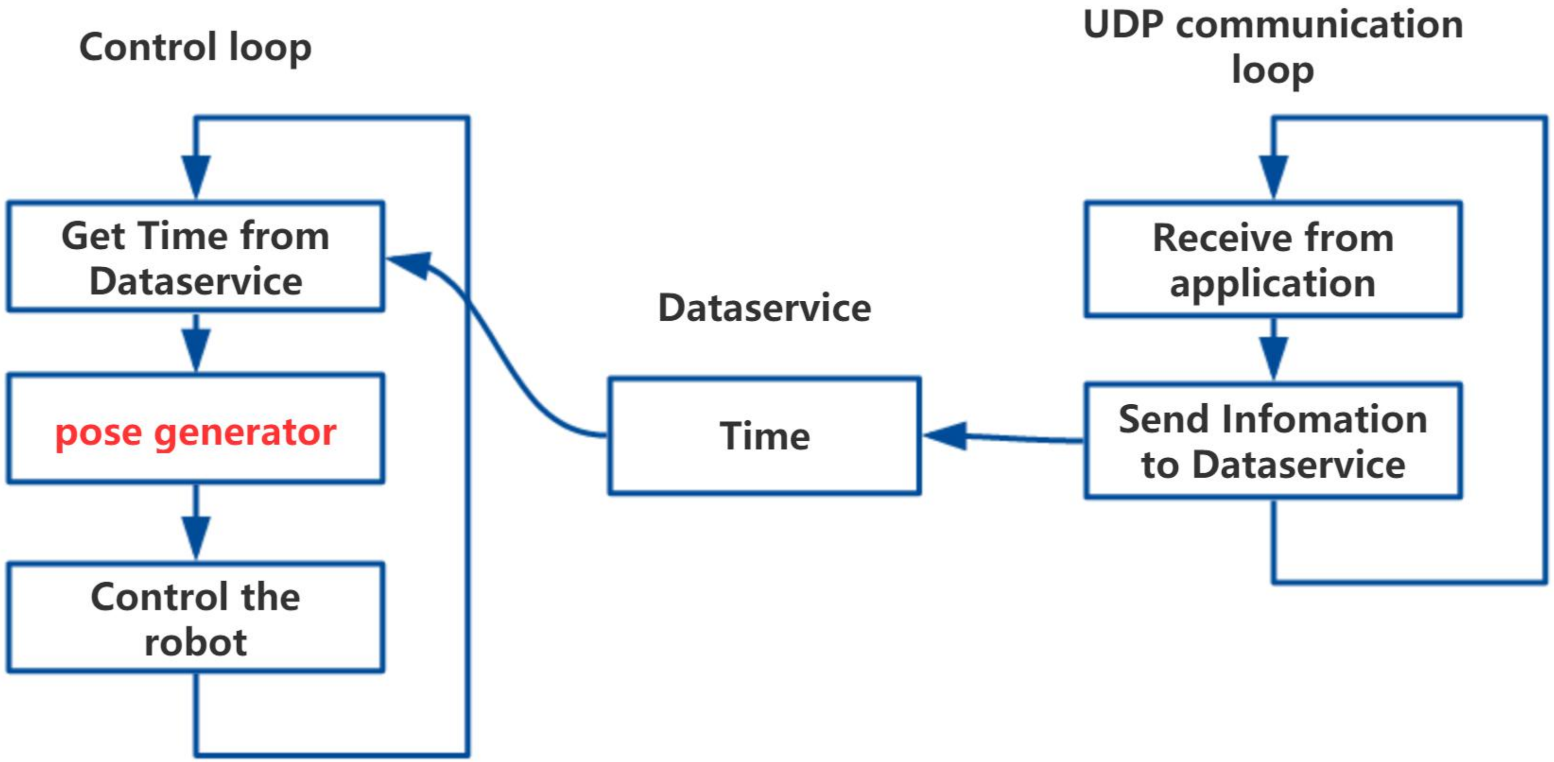
— The delays sometimes have exceeded 80us for a while (more than 20 cycle), which may cause the error.



3 UDP Communication
3.5 The asynchronous UDP Communication - 3

3.5.1 The structure of the asynchronous UDP Communication - 3

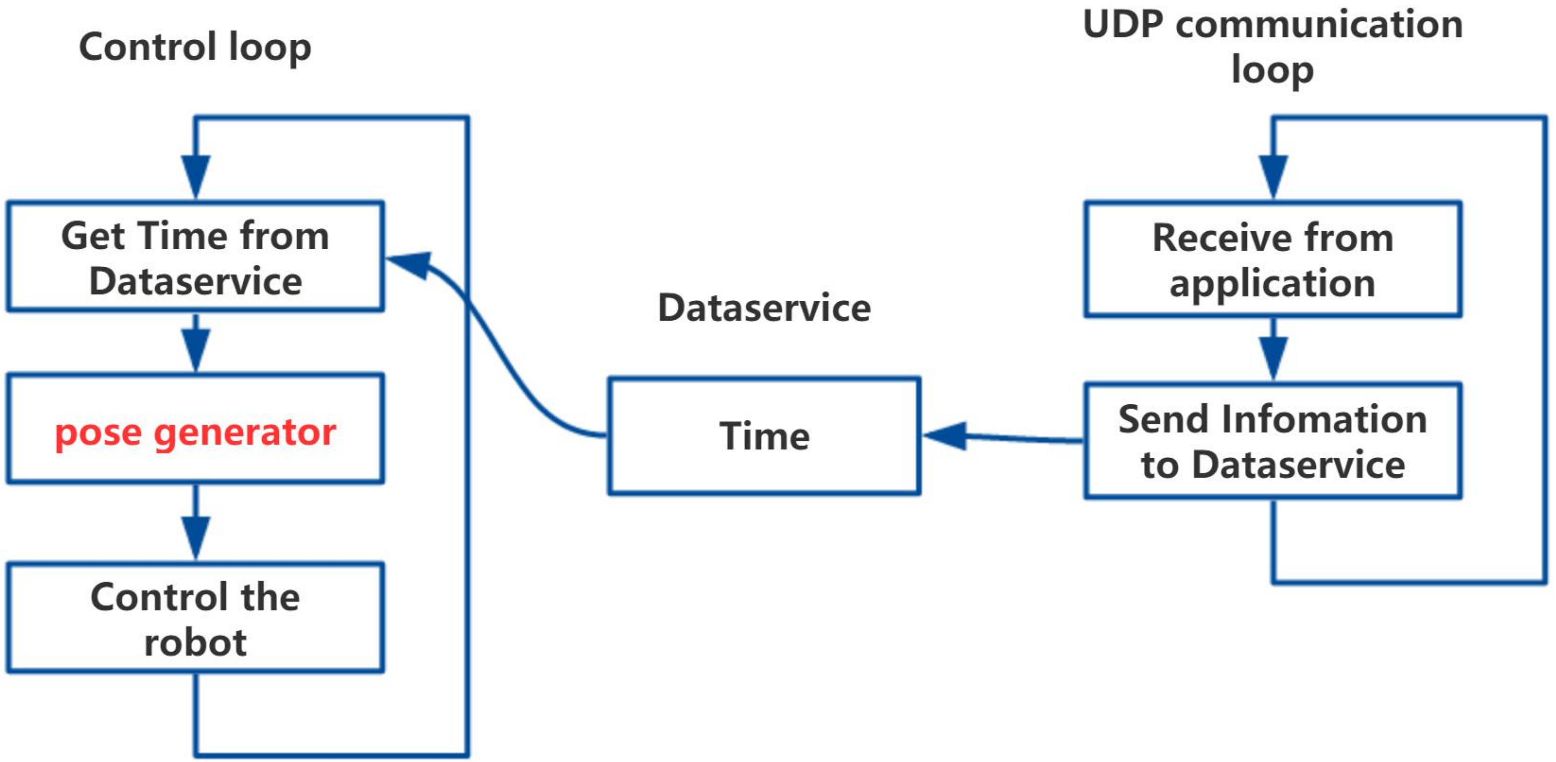
— The structure of the interface program



3 UDP Communication
3.5 The asynchronous UDP Communication - 3

3.5.1 The structure of the asynchronous UDP Communication - 3

— The structure of the interface program



Advantage:

small latency in the control loop

Disadvatage:

large latency of the data through the UDP

Performance:

Similar to the asynchronous UDP Communication - 1

3 UDP Communication
3.5 The asynchronous UDP Communication - 3

3.5.1 The structure of the asynchronous UDP Communication - 3

```
— robot.control([&time, &DS](const franka::RobotState &robot_state,
—                            franka::Duration period) -> franka::CartesianPose {
—     time_period = period.toSec();
—     command_time=DS.RecvCommand();
—     TimeGenerator(current_time, command_time, time_period);
—     double angle_x = M_PI / 4 * (1 - std::cos(M_PI / 5.0 * current_time));
—     double delta_x = kRadius * std::sin(angle_x);
—     std::array<double, 16> new_pose = initial_pose;
—     new_pose[12] += delta_x;
—     new_pose[14] += delta_z;
—     DS.UpdateSendData(current_time,robot_state.O_T_EE_c);
—     return new_pose;
—   });
```

3 UDP Communication

3.5 The asynchronous UDP Communication - 3

3.5.1 The structure of the asynchronous UDP Communication - 3

```
void TimeGenerator( double &current_time, double target_time, double period )
{
    double time_interval = target_time - current_time;
    if( time_interval > period )
    {
        current_time += period;
    }
    else if( -time_interval > period )
    {
        current_time -= period;
    }
}
```

3 UDP Communication
3.5 The asynchronous UDP Communication - 3

3.5.1 The Results

— This may be the only program that can “run” with the robot.

— Problem is that

— 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.

— **Why always the acceleration ?!!**

4 Problem Discussion
To find out the acceleration problem

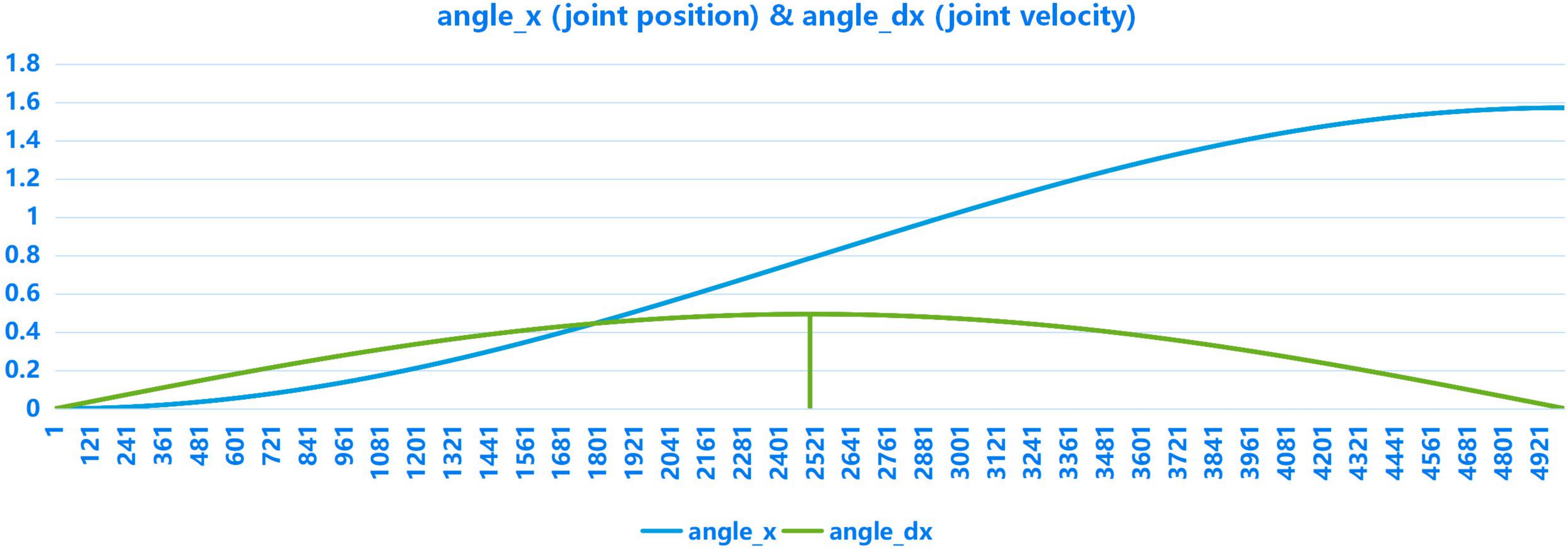
4.1 Simulate the change of the robot's motion parameters

- The value of period.toSec() is specially defined as 0.001, all the initial values are 0

```
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;  //          time+=0.001; - ideal case
    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;
}
```

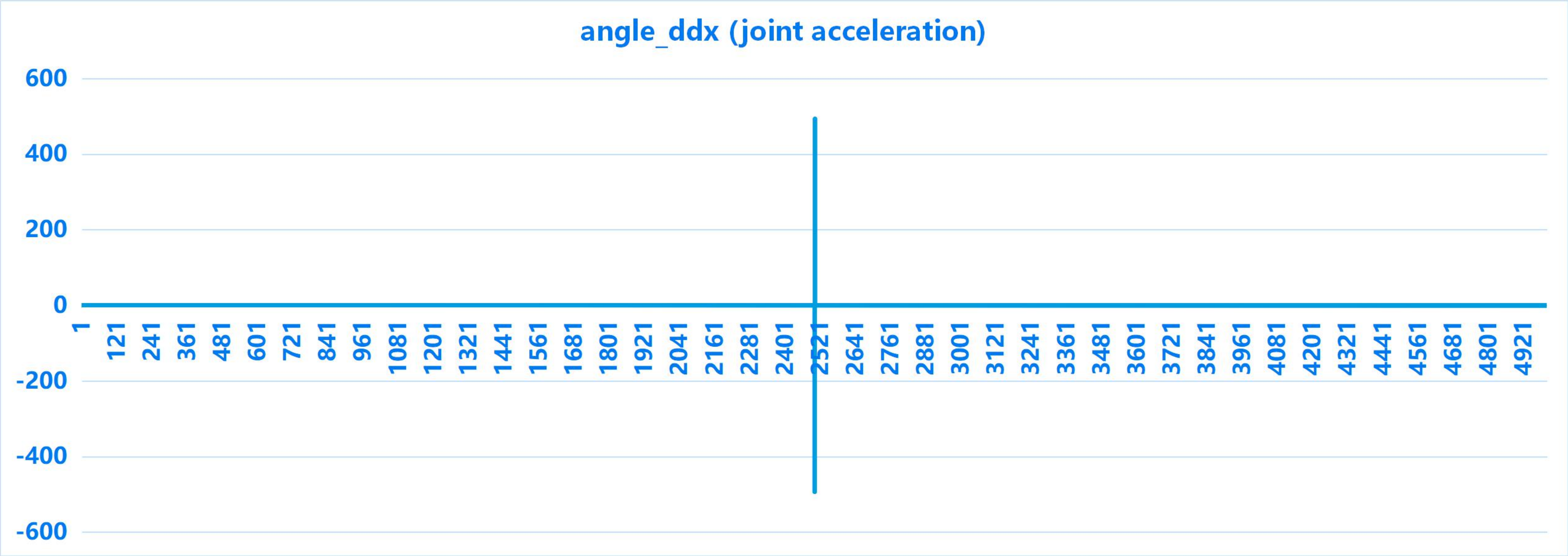
4 Problem Discussion
To find out the acceleration problem

4.3 Change of the robot's motion parameters - lost a command in one control loop



4 Problem Discussion
To find out the acceleration problem

4.2 Change of the robot's motion parameters - lost a command in one control loop

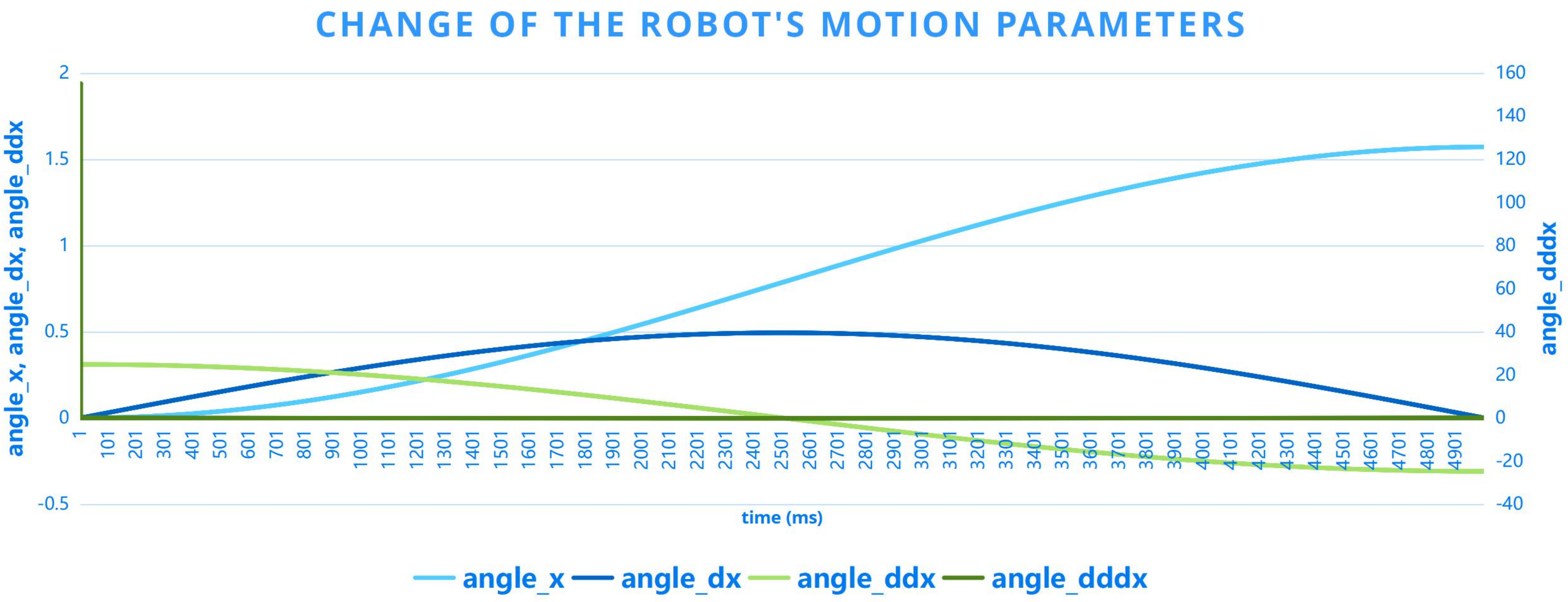


4 Problem Discussion
To find out the acceleration problem

4.2 Change of the robot's motion parameters - lost a command in one control loop

4 Problem Discussion
To find out the acceleration problem

4.3 Change of the robot's motion parameters - ideal case



4 Problem Discussion
To find out the acceleration problem

4.4 Compare to the limit of the official website [7]
— The results in the 4.2 greatly exceed the limit

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

5 Conclusions
Of reducing the Traffic in the Control Loop for the Robot Arm

5.1 Three Tips for the UDP communication in the interface application

— 1. A synchronous UDP Communication because of its unstable and delay could not be directly used in the robot control loop.

— 2. 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.

5 Conclusions
Of reducing the Traffic in the Control Loop for the Robot Arm

5.2 Solution for acceleration probem

- **In the example:**
- time += period.toSec();
- double angle_x = M_PI / 4 * (1 - std::cos(M_PI / 5.0 * time));

- **Now, the new pose should be calculated by the velocity instead of the time**

- velocity = function(v(x-2), v(x-1), desired angle from application);
- double angle_x += period.toSec() * velocity;

- where v(x-2), v(x-1) is the previous velocity, and the function will according to the limit and the desired angle to calculate the optimal speed

5 Conclusions
Of reducing the Traffic in the Control Loop for the Robot Arm

5.3 Solution for acceleration probem

- **In the example:**
- time += period.toSec();
- double angle_x = M_PI / 4 * (1 - std::cos(M_PI / 5.0 * time));

- **Now, the new pose should be calculated by the velocity instead of the time**

- velocity = function(v(x-2), v(x-1), desired angle from application);
- double angle_x += period.toSec() * velocity;

- where v(x-2), v(x-1) is the previous velocity, and the function will according to the limit and the desired angle to calculate the optimal speed

References
Reducing the Traffic in the Control Loop for the Robot Arm

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

[2] Christopher M. Kohlhoff, Daytime.4 - A synchronous UDP daytime client, Retrieved 26 February 2019, from : https://www.boost.org/doc/libs/1_35_0/doc/html/boost_asio/tutorial/tutdaytime4.html

[3] Programming Interfaces Guide - Chapter 8 Socket Interfaces - Socket Basics, Oracle Corporation, Retrieved 26 February 2019, from : https://docs.oracle.com/cd/E19120-01/open.solaris/817-4415/sockets-18552/index.html

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

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

[6] 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/

References
Reducing the Traffic in the Control Loop for the Robot Arm

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

Thank you!