Lappeenranta University of Technology
Faculty of Technology Management
Degree Program in Information Technology

Bachelor's Thesis

Petteri Pekonen

DESIGN AND IMPLEMENTATION OF ROBOT GRIPPER INTERFACE

Examiners: Jarmo Ilonen D.Sc. (Tech.)

Supervisor: Jarmo Ilonen D.Sc. (Tech.)

ABSTRACT

Lappeenranta University of Technology
Faculty of Technology Management
Degree Program in Information Technology

Petteri Pekonen

Design and implementation of robot gripper interface

Bachelor's Thesis

2012

36 pages, 15 figures, 4 table, and 3 appendices .

Examiners: Jarmo Ilonen D.Sc. (Tech.)

Keywords: Interface, Robotics, Grasping, Gripper, Robotiq

This thesis presents a design for an asynchronous interface to Robotiq adaptive gripper s-model. Designed interface is a communication layer that works on top of modbus layer. The design contains function definitions, finite state machine and exceptions. The design was not fully implemented but enough was so that it can be used. The implementation was done with c++ in linux environment. Additionally to the implementation a simple demo program was made to show the interface is used. Also grippers closing speed and force were measured. There is also a brief introduction into robotics and robot grasping.

TIIVISTELMÄ

Lappeenrannan teknillinen yliopisto
Teknistaloudellinen tiedekunta
Tietotekniikan koulutusohjelma

Petteri Pekonen

Robottikäden ohjelmointirajapinnan suunnittelu ja toteutus

Kanditaatintyö

2012

36 sivua, 15 kuvaa, 4 taulukko ja 3 liitettä .

Tarkastajat: TkT Jarmo Ilonen

Hakusanat: Käyttöliitymä, Robotiikka, Tartunta, Tartuin, Robotiq

Tässä työssä esitellään asynkrooninen käyttöliittymä Robotiq adaptive gripper s-model robottikäteen. Käyttöliittymä on kommunikaatiokerros, joka toimii modbus-kerroksen päällä. Suunnitelma sisältää funktiomäärittelyt, tilakoneen ja poikkeukset. Suunnitelmasta toteutettiin tarpeeksi, että käyttöliittymää voidaan käyttää. Toteutus tehtiin c++ kielellä linux-ympäristössä. Toteutuksen lisäksi yksinkertainen demo tehtiin näyttämään, miten käyttöliittymää käytetään. Näiden lisäksi Käden sulkemisnopeus ja -voima mitattiin. Työ sisältää myös lyhyen esittelyn robotiikasta ja robottitarttumisesta.

CONTENTS

Appendix 3: gripper registers

ABBREVIATIONS AND SYMBOLS

ms	millisecond
API	Application Programming Interface
FSM	Finite State Machine

1 INTRODUCTION

This bachelors thesis was done in department of information technology in Lappeenranta university of technology. The goal of this thesis was to design and implement a robot gripper interface for Robotiq adaptive gripper S-model.

1.1 Backround

Department of information technology in Lappeenranta university of technology has obtained a new robot gripper. It has been obtained because department does robot grasping and manipulation research. Department has special interest in sensor based robot manipulation research. In sensor based manipulation one wants to be able to adjust speeds and forces as much as possible. This way it is possible to grasp weak objects and improve the grip during grasping. New gripper gives possibility to this but the only interface is modbus interface. To make use of gripper easier a new interface is needed. This new interface will hide modbus interface and simplifies the use of gripper.

1.2 Objectives and Restrictions

The objective of this thesis is to design and implement an interface to the robotiq adaptive gripper. The interface will become part of the open source library itlabcpp. The definitions of usable functions of the interface will be done in cooperation with few future users. Programming will be done with c++ programming language in linux environment. The interface will be asynchronous so that even in error situations it will not block the program using it. The interface will use modbus protocol to communicate with the gripper. In this case modbus works on top of ethernet. The gripper updates its status every 5ms so the interface should also retrieve gripper status info every 5ms. Also some kind of demo program will be made to show how the interface is used.

The interface uses values between 0 and 255 to indicate speed and force. The relationship between these values and physical quantities is not known. There for these relationships need to be defined by measurements.

1.3 Structure of the Thesis

As a theory part section 2 contains short introduction into robotics and robot grasping. It also introduces core concepts and the gripper that is being used. In section 3 the design of the interface layer is presented. Section 4 contains what of the design was implemented and how public functions were implemented. Section 5 contains details about the demonstration program. Section 6 contains the speed and force measurements that were made as a part of this thesis. Section 7 has discussion and Section 8 has conclusions.

2 THEORY

This chapter provides a brief introduction into robotics and robot grasping. It also introduces the used gripper and defines some core concepts in this thesis.

2.1 Robotics

The term robot comes from a play "Rossum's Universal Robots" written by Czech playwright Karel Čapek in 1920. But the research field of robotics did not come about until the middle of twentieth century. That is when there were advances in research of artificial intelligence, mechanics, controls, computers and electronics. The first computer controlled robots were made in the mid-to-late twentieth century after the development of integrated circuits, digital computers and miniaturized components. These were industrial robots.In the 1980s a new kind of robots were made. These were mobile robots that had some way to move around and sense the enviroment. In todays world robots are used in many places. These places include industrial manufacturing, healthcare, transportation and exploration of deep space and sea.[1]

A robotics researcher has to have a understanding of many different topics. These fundamental topics include kinematics, dynamics, mechanical design and actuation, sensing and estimation, motion planning, motion control, force control, robotic system architectures and programming and reasoning methods for task planning and learning.[1]

2.2 Grasping

Human hand has three main functions. Those are to explore, to restrain objects and to manipulate objects [2]. The first one is researched under topic of haptics. Robot grasping on the other hand is attempting to understand and emulate the latter two. The work of Asada and Hanafusa [3] and Salisbury's first attempts to create three fingered robotic hand [4] can be considered the start of the field of robot grasping [2].

There is two types of grasping. A fingertip grasping and an enveloping grasping [5]. In fingertip grasping distal phalanges are used for grasping. This is usually used for finer manipulation. An enveloping grasp is where fingers surround the object. An enveloping grasp is more stable and can grasp heavier objects [6].

There is also two closure types. These are force closure and form closure. The grasp is force closure if contacts can exert any force and moment and that the objects motion is resisted [7]. Form closure means that grasp can resist any external force and moment applied to grasped object [8].

There is many other research areas in robot grasping. One of those is force analysis to choose grasp forces to minimize slippage [2]. This is closely related to grasp stability research. There is also research into hand dynamics and kinematics [2]. There is also some research into grasping multiple objects at the same time [9]. Grasp performance measurements have also been developed to help in grasp planning [10].

2.3 The gripper

The gripper used is three fingered robotiq adaptive gripper[Fig 1] s-model. Each finger has three joints. The gripper can automatically adapt to the shape of an object it is grasping. The closing speed and force can be adjusted. It is also possible to do partial closing or opening with the gripper. The gripper also sends feedback from grasping.

Figure 1. Robotiq adaptive gripper s-model[11].

The gripper has four build-in modes[Fig 2]. These modes are basic mode, wide mode, pinch mode and scissor mode. In basic mode the gripper closes by moving all three fingers so that finger a goes between fingers b and c. This allows grasping of different kinds of objects. Wide mode is almost same as basic mode but fingers b and c are as far away from each other as possible. In pinch mode fingers b and c are very near of each other. This makes encompassing the object impossible, because grasping is done with finger tips. This allows picking up small objects that have to be picked precisely. In scissor mode objects are picked with only fingers b and c. This is done by moving them laterally. This mode is not very powerful but it is precise.

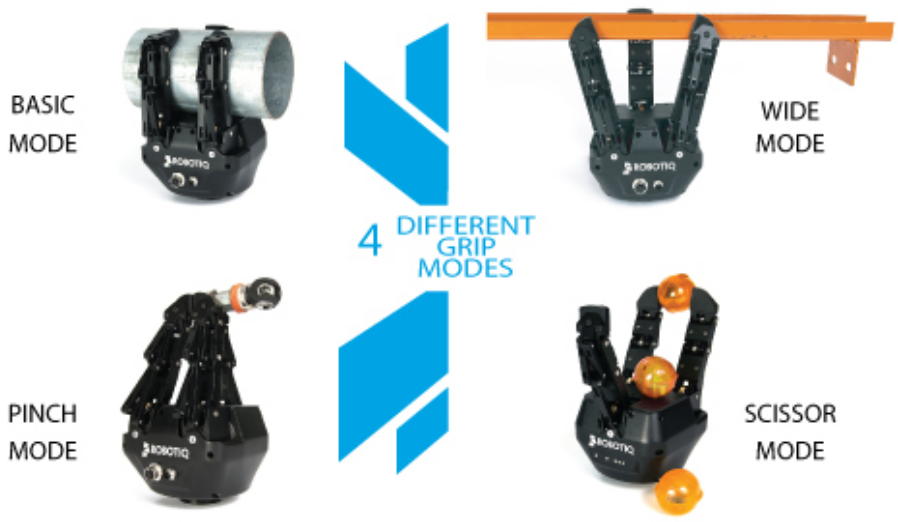

Figure 2. The four build-in modes of the gripper[11].

It is also possible not to use any of the four build-in modes but to use individual finger control. This allows opening and closing of each finger with out moving other fingers. It also allows to set the speed and force for each finger individually.

2.4 What is interface/ definition of interface

In this work interface refers to an API(Application Programming Interface). Interface is a set on function, constants and rules that do not change. Interface provides easier to use functions to a given resource than using it directly. Interface also provides a layer of abstraction so that if the given resource is changed the implementation can also be changed so that programs using the interface are not effected in anyway.

3 DESIGN

The API(Application Programming Interface) function definitions were done in cooperation with Ville Kyrki, Jarmo Ilonen and Janne Laaksonen. It was decided that only individual finger control would be used instead of built-in modes. We did not want to create separate functions for each finger. For this reason we decided to use integer constants to tell which finger is being given a command. Then we wanted to be able to give commands to multiple fingers at the same time so constant were designed to work in mask. It was also decided that this interface would follow a singleton design pattern [12]. This means that there can be only a single interface object within a single program.

The interface layers structure[Fig 3] contains a separate communication thread [13], because of the requirement that none of the functions may block. In this structure API functions change shared data. Communication thread then check this shared data and communicates with the gripper according to commands. Communication thread then also changes the shared data to give response.

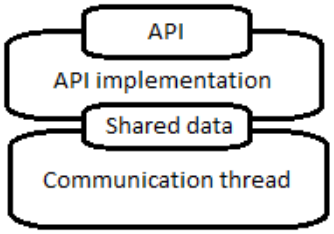

Figure 3. The structure of the interface layer.

3.1 Protocol stack

From the protocol stack[Fig 4] we can see that this interface is used to manipulate the grippers registers. A modbus protocol, which is an industrial standard[14], is used for actual data transfer. In this work free libmodbus is used as modbus implementation [15].

3.2 State machine

The picture[Fig 5] contains the full FSM(finite state machine) diagram of the interface. Not all of these states have been implemented. To make state machine diagram easier

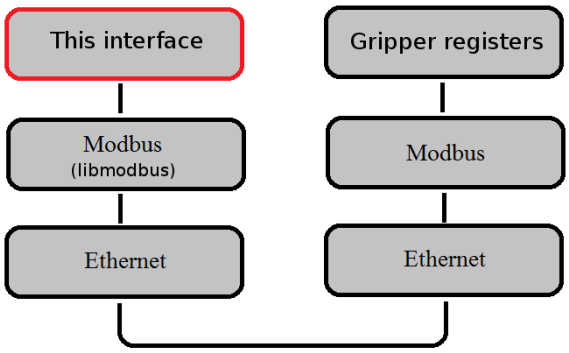

Figure 4. The protocol stack.

to read non-standard "any state" state symbol has been used instead of drawing an arrow from every state.

The FSM contains few additional do states because of the requirement that functions may not block. This is because the actual message sending and receiving is done in separate thread. So these do states indicate for this thread that it should do something. For example it should send some particular message.

disconnected: This state indicates that there is no connection to the gripper. This can be because no connection has been formed or existing connection was lost. This is also starting state.

do_activation: This state indicates that the interface should send activation signal to the gripper.

wf_activation: This state indicates that the interface is waiting for the gripper to finish it's activation routine.

running_idle: This state indicates that is running normally and accepts set commands.

running_cc: This state is basically same as running_idle but indicates that a command has been given and should be send to the gripper.

fault: This state means that the gripper is giving a fault status.

do_atr: This state indicates that the interface should send automatic release signal to the gripper.

wf_atr: This state indicates that the interface is waiting for the gripper to finish automatic

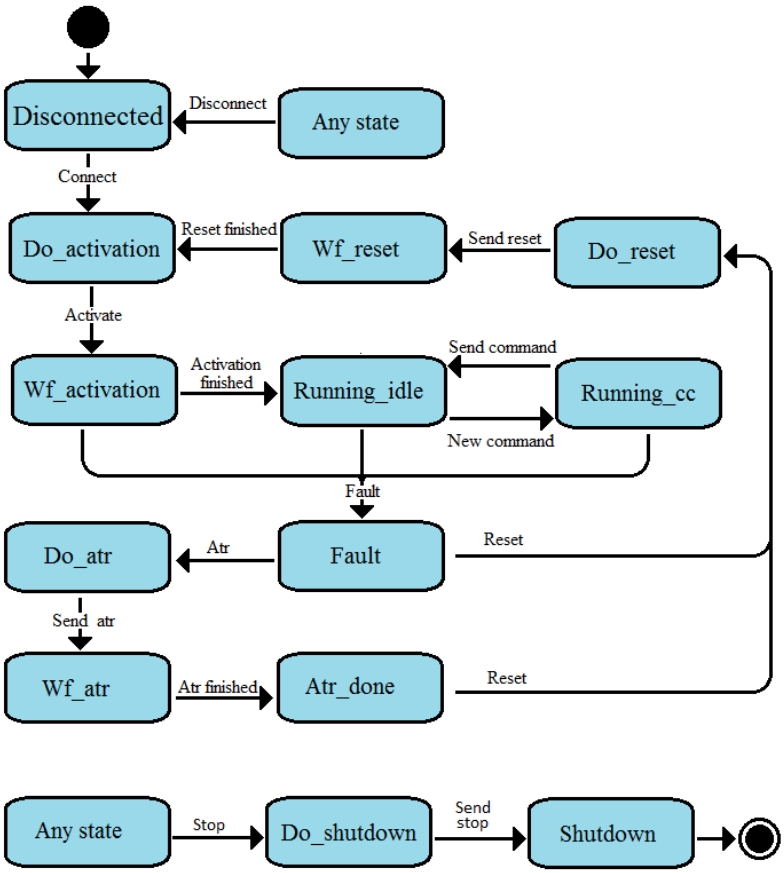

Figure 5. Full designed FSM.

release.

atr_done: This state indicates that automatic release has been completed.

do_reset: This state indicates that the interface should send reset signal to the gripper.

wf_reset: This state indicates that the interface is waiting for the gripper to finish reset.

do_shutdown: This state indicates that the interface should do shutdown routine. Once shutdown routine is finished state is changed to shutdown. In this state the interface does not accept any commands.

shutdown: This indicates that the interface has been shutdown.

3.3 interface/functions

The interface has two classes that the user will use. The RobotiqHand class and the RobotiqStatus class. RobotiqHand class is used to control the gripper. RobotiqStatus class contains grippers status information.

3.3.1 RobotiqHand class

RobotiqHand class is the main class of the interface. It handles user commands and delivers them to the gripper. This class contains the FSM and a separate thread, that handles communication between the gripper and the interface. Following tables contains all designed public functions[Tab 1] and constants[Tab 2] of the RobotiqHand class.

3.3.2 RobotiqStatus class

RobotiqStatus class contains gripper status information. It also contains functions and constants to interpret the status information more easily. Following tables contains all designed public functions[Tab 3] and constants[Tab 4] of the RobotiqStatus class.

Table 1. Interface functions.

Function name	**Parameters**	**Description**
setPosition	int pos, int mask	moves the fingers given in mask to given position.
setSpeed	int speed, int mask	sets the moving speed of the fingers given in mask.
setForce	int force, int mask	sets the moving force of the fingers given in mask.
isActivated		checks if gripper is activated and ready for commands.
isReset		checks if gripper is in reset state.
isMoving		checks if gripper has finished moving command.
getStatus		Returns most recent RobotiqStatus object.
atr		Does automatic release. This is only allowed in fault state.
reset		Resets the gripper.
start		Initializes gripper interface. This has to be called before interface can be used
stop		Stops the gripper and internal thread.

Table 2. Interface constant.

Constant name	**Value**	**Description**
FINGERA	0x1	This is used in mask of set functions for finger A.
FINGERA	0x2	This is used in mask of set functions for finger B.
FINGERA	0x4	This is used in mask of set functions for finger C.
SCISSOR	0x8	This is used in mask of set functions for scissor.
OPEN	0	Constant for setPosition function for opening gripper.
CLOSED	255	Constant for setPosition function for closing gripper.
WIDTH_NORMAL	100	Meant to used in setPosition function to move scissor.
WIDTH_WIDE	0	Meant to used in setPosition function to move scissor.
WIDTH_PINCH	255	Meant to used in setPosition function to move scissor.
SPEED_MAX	255	Maximum speed for setSpeed function.
SPEED_MIN	0	Minimum speed for setSpeed function.
FORCE_MAX	255	Maximum force for setForce function.
FORCE_MIN	0	Minimum force for setForce function.

Table 3. Status functions.

Function name	Parameters	Description
getFingerStatus	int finger	Has finger stopped due to a contact.
getFingerPos	int finger	Current position of a finger.
getFingerReqPos	int finger	Requested position of a finger.
getFingerForce	int finger	Electric current used at the moment.
getFaultStatus		Use fault constants for interpretation
isActivated		checks if gripper is activated and ready for commands.
isReset		checks if gripper is in reset state.
isMoving		Use the one from RobotiqHand class.

Table 4. Status constant.

Constant name	Value	Description
FINGERA	0x1	Finger A constant for get functions.
FINGERA	0x2	Finger B constant for get functions.
FINGERA	0x4	Finger C constant for get functions.
SCISSOR	0x8	Scissor constant for get functions.
FINGER_IN_MOTION	0x00	
FINGER_CONTACT_OPENING	0x02	finger has stopped due to contact while opening.
FINGER_CONTACT_CLOSING	0x01	finger has stopped due to contact while closing.
FINGER_AT_POS	0x03	finger is at requested position.
FAULT_NONE	0x00	finger is at requested position.
FAULT_DELAYED_ACTIVATION	0x05	Activation must be completed prior to action.
FAULT_DELAYED_MODE_CHANGE	0x06	Mode change must be completed prior to action.
FAULT_ACTIVATION_NEEDED	0x07	The activation bit must be set prior to action.
FAULT_COM_CHIP_NOT_READY	0x09	Mode change must be completed prior to action.
FAULT_CHANGE_MODE_MINOR	0x0A	interferences detected on Scissor (under 20 sec).
FAULT_ATR	0x0B	Automatic release in progress.
FAULT_ACTIVATION	0x0D	Activation fault, verify that no interference or other error occured.
FAULT_CHANGE_MODE_MAJOR	0x0E	interferences detected on Scissor (over 20 sec).
FAULT_ATR_COMPLETE	0x0F	Automatic release completed. Reset and activation is required.

3.4 Exceptions

The interface can throw exceptions[Fig 6] when there is an error that requires an action from the user. The user can be a program. All of these exception are derived from the RobotiqException base exception class. There are two main branches of exceptions. These are state exceptions and connection exceptions. Both branches have few more specific exceptions.

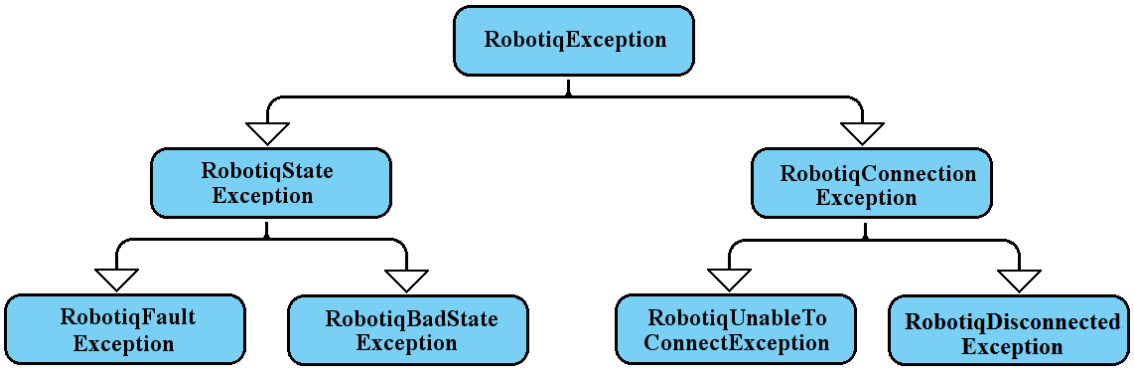

Figure 6. The inheritance hierarchy of exception classes.

4 IMPLEMENTATION

Not everything from design were implemented. This can be seen in the implemented FSM[Fig 7] as it does not contain states for atr and reset functionality. These were left out of implementation because they were deemed unnecessary for now.

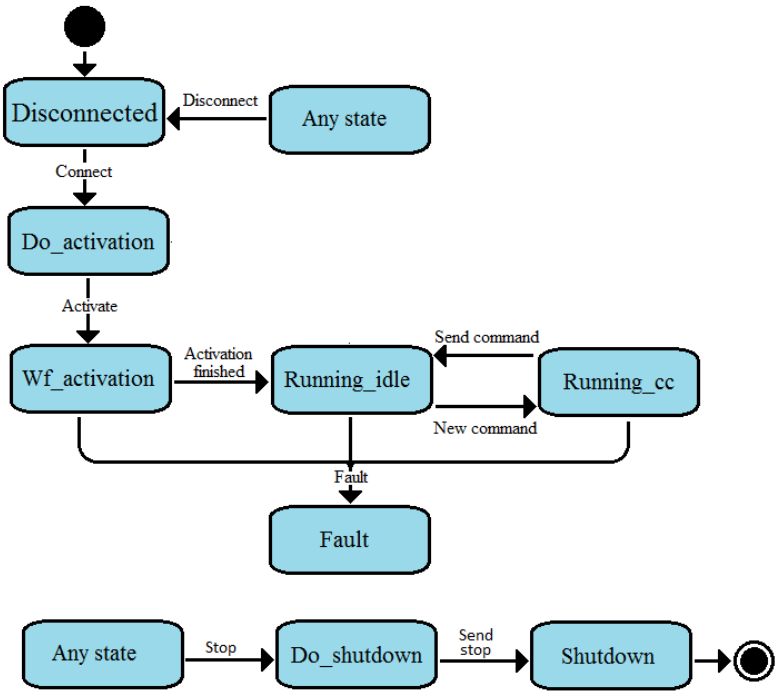

Figure 7. Implemented FSM.

4.1 RobotiqHand class

This section explains how some of the functions of the Robotiqhand class were implemented. Atr and reset functions were not implemented. Appendix[App 3] contains register mapping, but not full explanation of them.

set functions: These functions update interface layers internal copy of the grippers register. After this update it changes state so that the updated register is send to the gripper.

isActivated: This function return true if register gIMC is 0b11. This means that activation or mode change is completed.

isReset:This function return true if register gIMC is 0b00. This means that the gripper is in reset (or automatic release) state. Reset state needs activation before gripper can be used again.

isMoving: This function returns true if the most recent Roboticstatus objects ismoving function returns true. Because there is some delay between giving move command and sending the command, this function also return true if command has not been send. Command delivery is checked by two things. First if current state is running_cc meaning that commands need to be sended. Second by comparing internal copy of requested positions to the most recent Roboticstatus objects requested positions.

start: At first this function creates a connection to the gripper. Then the gripper is activated by sending rACT bit. At the same time individual finger and scissor control is enabled by rICF and rICS bits and the gripper is stopped by rGTO bit, which does not stop activation. The function then waits for activation to finish. After that it starts the communication thread and waits for it to start.

stop: This function changes current state to do_shutdown and waits for the communication thread to end. The communication thread ends after stopping the gripper by rGTO bit.

4.2 RobotiqStatus class

This section descripes how functions of the RobotiqStatus class work. Appendix[App 3] contains register mapping, but not full explanation of them.

get finger functions: These return appropriate register values directly.

getFaultStatus: This function returns gFLT bits directly. Compare this to given constants to get details about the fault.

isMoving: This function checks if gSTA is 0b00. This means that the gripper is in motion towards requested position.

isActivated:This function checks if gIMC is 0b11. This means that activation and mode change are completed.

isReset:This function checks if gIMC is 0b00. This means that the gripper is in reset (or automatic release) state. See fault status if the gripper is activated.

5 DEMO

As a part of this thesis a demo program was made. The purpose of this demo is to show how interface is used. Because of this purpose a the demo was designed so that it was informative for programmers instead of showy movements.

The demo program has four parts. These parts are basic open and close movements, changing speed, changing force and status information. Basic open and close movements contains pinch, wide, normal and scissor movements. Changing speed shows how to change speed for individual fingers or multiple fingers at the same time. Changing force is basically same as speed but force is changed instead of speed. Status information shows how to check finger status and how to get current data that translates to force. It also shows how to check fault state.

6 MEASUREMENTS

The interface uses values between 0 and 255 for speeds and forces. It is part of this thesis to measure what these values are in physical quantity.

6.1 Speed

The time between giving close command to interface and interfaces ismoving function returning false was measured. This gives a close approximate of grippers closing speed for given speed value. The value is not exact because it includes network delay and maybe small delay before interface sends the command. Measurements were made in order from slowest to fastest.

Because these measurement include small errors three runs were made. Each run measures values starting from zero and every fourth value from there on. The picture [Fig 8] contains a plot of average values from these three measuring runs. Results show that speed is not linear. Instead the speed increase decreases with higher values.

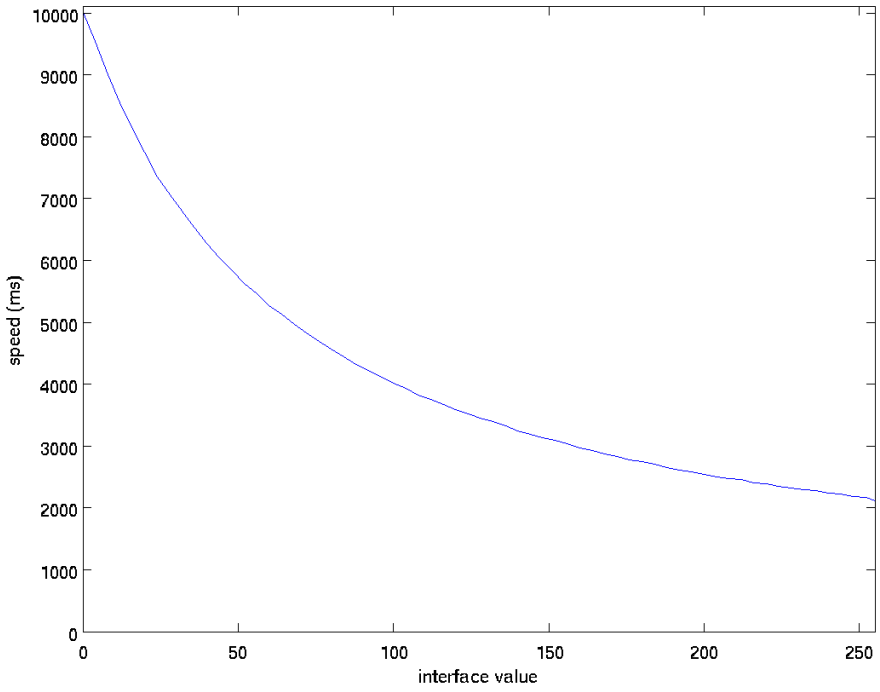

Figure 8. Graph of gripper's closing speed.

6.2 Force

Force was measured by grasping a scale with different force values. Scale measures mass from the force gravity causes to it. Therefor scale can also be used to measure force. Equation 1 was used to convert scale reading to force.

$$F = mg \tag{1}$$

Where m is the value read from scale and g is $9.81m/s^2$.

To eliminate the effect of gravity scale was mounted vertically[Fig 9]. The gripper then grasped it from above. After gripper had stopped value from scale was read manually. This produced some errors to data because value on scale always dropped after gripper had stopped. For this reason values from scale were recorded with varying precision. Sometimes first seen hundred was recorded sometimes first ten. The scale used has maximum value at a bit over 3000 grams. Because of this sometimes scale gave errors during measurements.

Figure 9. Force measuring setup.

Three measuring runs were made. Each run measured every 25th value starting from zero. Values were measured starting from zero and going up by 50 and then starting from 225 and going down by 50. The picture [Fig 10] contains a plot of average values from these three measuring runs. Results seemed to be almost same for every force value. There were few random low values.

Because this setup produced unreliable results it was improved by placing a piece of foam between the scale and fingers[Fig 11]. This allows more time and space for gradual increase of force.

Three measuring runs were made in same way as before. This time the values from the scale[Fig 12] were much more as expected. They were generally increasing as force value

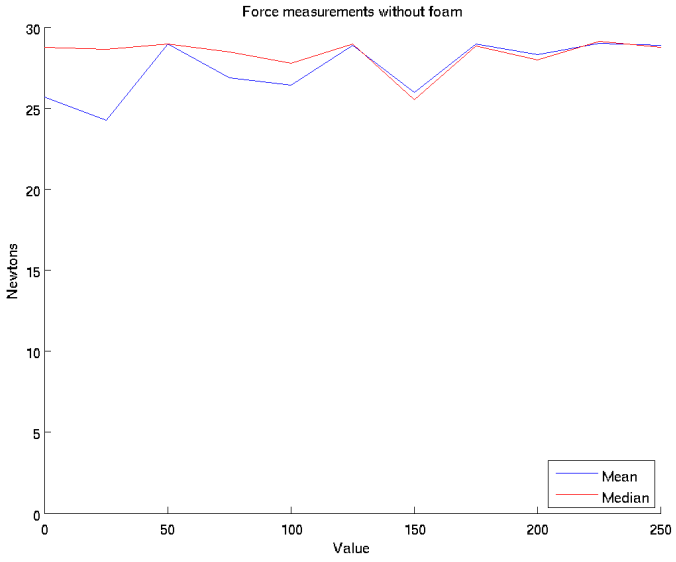

Figure 10. Force measurements.

Figure 11. Force measuring setup with foam.

was increasing. Even though result were more reliable than on previous setup there was still sometimes large random variations.

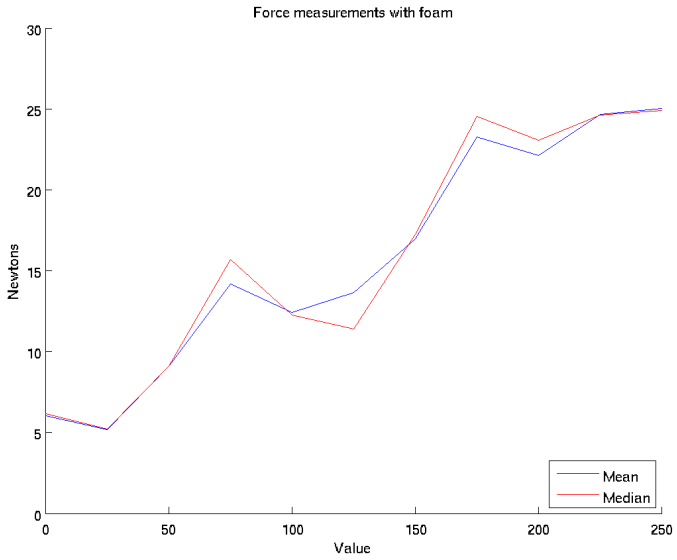

Figure 12. Force measurements with foam.

Because measurements seemed to be unreliable, currents were also recorded. Maximum currents from a run without a foam[Fig 13] show a relation between maximum current and force. Maximum Currents from a run with a foam[Fig 14] show a relation between maximum currents of fingers b and c and force. This is because fingers b and c were pressing the foam.

Timelines of currents with and without foam[Fig 15] show why results were better with foam. Without foam the currents peak almost instantly. Where as with foam currents have some time to reach their peaks thus allowing better results.

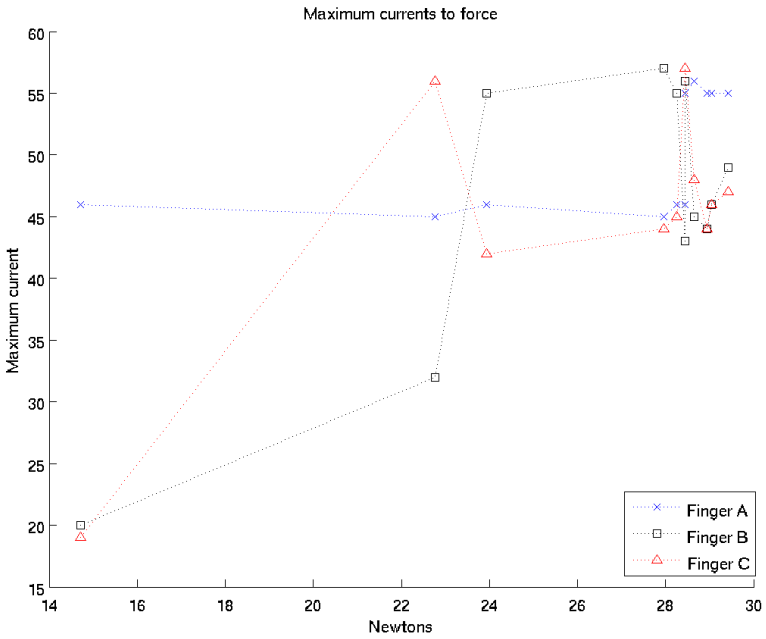

Figure 13. Relationship between current and force from second run without foam.

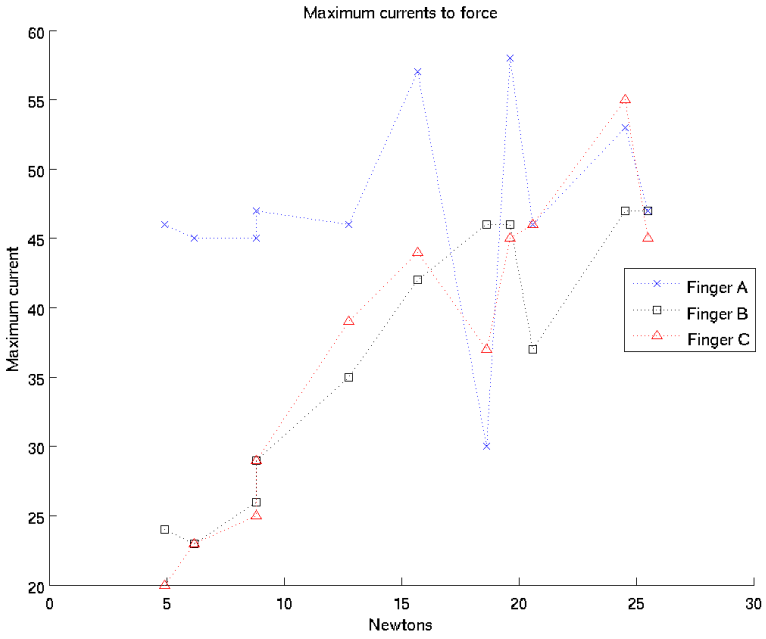

Figure 14. Relationship between current and force from first run with foam.

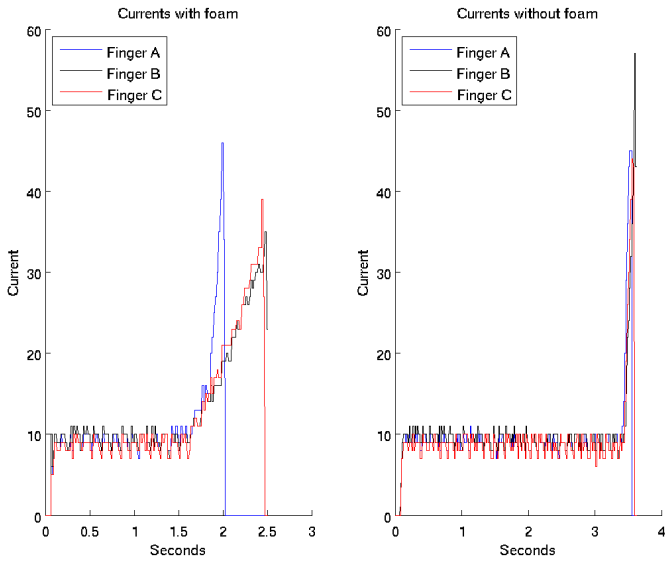

Figure 15. Timelines of current from a run with and without foam.

7 DISCUSSION

The two main goals of the interface were to hide register manipulation behind easy to use functions and to have non blocking functions. Non blocking functions were necessary because the interface would be used inside robot control program and could produce real danger if it would freeze the program. To Make the functions non blocking a separate communication thread was used. Thread was used because it is easiest method and libmodbus doesn't have asynchronous communication mode [15].

The four build-in modes were not used. This is because individual finger control was needed, because the gripper would be used in sensor based manipulation research. This requires the ability to adjust speed and force as much as possible. Changing build-in modes would also result in gripper movement that would not necessary be wanted. Because of this the build-in modes were simulated using individual finger control.

After deciding only to use individual finger control it was decided that we would use functions with a parameter to indicate which fingers it is are affected. This was done because we didn't want to have too many functions by having separate function for each finger.

The atr and reset parts of the fsm were left out of the implementation, because they are not needed in normal operations. The other reason was that they can be added later if needed.

7.1 Future Work

There is several things that were left for future work. First of all fault recovery. Some faults can be cleared without resetting. This leads to the second thing, which is reset. Also automatic release functionality was left for future work, because it was not needed yet. A possibility to reconnect should be made. With reconnect one should also think if interface should try reconnecting automatically. It is also necessary to think if interface should perform activation after reconnecting, especially in automatic reconnect.

8 CONCLUSIONS

Because the interface was required to be asynchronous, the state machine has some extra states. These extra states are used in communication between API and communication thead. The parts of the interface that were implemented are working. The parts that weren't implemented were parts that are not needed in normal operations. Implementation is now part of the itlabcpp library.

Measuring speed was rather simple and it went well. measuring force turned out to be a bit tricky. A scale was used to measure the grasping force. The problem was that the value on scale started to drop as soon as gripper stopped. Also softness had to be added to the scale to get any kind of meaningful results.

REFERENCES

[1] Bruno Siciliano and Oussama Khatib. *Handbook of Robotics*, chapter 1 Introduction, pages 1–4. Springer, 2008.

[2] Antonio Bicchi and Vijay Kumar. Robotic grasping and contanct: A review. In *IEEE International Conference on Robotics and Automation*, volume 1, pages 348–353, 2000.

[3] H. Asada. *Studies on prehension and handling by robot hands with elastic fingers*. Ph.d. dissertation, Kyoto University, Kyoto, Japan, Apr 1979.

[4] Mason M.T and Salisbury J.K. Jr. *Robot hands and the mechanics of manipulation*. The MIT Press, Cambridge, MA, USA, 1985.

[5] Jeffrey Coates Trinkle. *The Mechanics and Panning of Enveloping Grasps*. a dissertation, University of Pennsylvania, Jan 1987.

[6] Makoto Kaneko, Yutaka Hino, and Toshio Tsuji. On three phases for achieving enveloping grasps - inspired by human grasping. In *IEEE International Conference on Robotics and Automation*, pages 385–390, 1997.

[7] Van-Duc Nguyen. Constructing force-closure grasps. In *IEEE International Conference on Robotics and Automation.*, volume 3, pages 1368–1373, 1986.

[8] Yun-Hui Liu. Computing n-finger form-closure grasps on polygonal objects. *The International Journal of Robotics Research*, 19(2):149–158, Feb 2000.

[9] Kensuke Harada and Makoto Kaneko. Enveloping grasp for multiple obj'ects. In *International Conference on Robotics & Automation*, volume 3, pages 2409 – 2415, 1998.

[10] Farzad Cheraghpour, S. Ali A. Moosavian, and Ali Nahvi. Multiple aspect grasp performance index for cooperative object manipulation tasks. In *International Conference on Advanced Intelligent Mechatronics*, pages 386 – 391, 2009.

[11] Robotiq inc. *Robotiq adaptive gripper instruction manual*, 111031 edition, 2011.

[12] Implementing the singleton pattern in c#, http://csharpindepth.com/articles/general/singleton.asp Retrieved 21.08.2012.

[13] Daniel Robbins. Posix threads explained, http://www.ibm.com/developerworks/linux/library/l-posix1/index.html, Retrieved 21.08.2012.

[14] Max Felser. Real-time ethernet - industry prospective. *PROCEEDINGS OF THE IEEE*, 93(6):1118–1129, June 2005.

[15] libmodbus, http://libmodbus.org, Retrieved 19.08.2012.

Appendix 1. Speed measurements

Table A1.1. Speed measurements in milliseconds.

Value	Run 1	Run 2	Run 3	Value	Run 1	Run 2	Run 3
0	10021	10021	10021	132	3402	3387	3397
4	9536	9536	9562	136	3326	3315	3326
8	9001	9031	8976	140	3239	3244	3261
12	8546	8546	8560	144	3201	3194	3194
16	8111	8116	8156	148	3154	3123	3143
20	7727	7733	7773	152	3083	3088	3104
24	7367	7344	7357	156	3027	3038	3042
28	7049	7080	7065	160	2967	2973	2982
32	6798	6792	6798	164	2901	2932	2947
36	6509	6536	6540	168	2865	2871	2885
40	6271	6266	6278	172	2850	2831	2811
44	6023	6049	6064	176	2770	2789	2760
48	5848	5807	5848	180	2740	2755	2750
52	5600	5655	5650	184	2720	2704	2704
56	5454	5458	5479	188	2648	2654	2674
60	5273	5281	5270	192	2612	2633	2609
64	5115	5130	5134	196	2588	2587	2569
68	4928	4984	4978	200	2542	2557	2548
72	4807	4841	4827	204	2513	2491	2502
76	4674	4695	4695	208	2473	2456	2502
80	4548	4574	4583	212	2462	2446	2452
84	4442	4443	4441	216	2395	2405	2427
88	4306	4336	4331	220	2390	2385	2406
92	4226	4225	4225	224	2344	2365	2350
96	4099	4118	4119	228	2314	2310	2339
100	3989	4017	4043	232	2295	2294	2299
104	3912	3942	3939	236	2280	2274	2274
108	3795	3831	3845	240	2238	2238	2239
112	3766	3765	3745	244	2218	2214	2233
116	3680	3675	3675	248	2183	2188	2193
120	3578	3593	3593	252	2173	2183	2162
124	3487	3539	3518	255	2097	2122	2137
128	3446	3453	3467				

(continues)

Appendix 1. Speed measurements

Appendix 2. Force measurements

Table A2.1. Force measurements with foam

value	Run 1(g)	Run 2(g)	Run 3(g)
0	630	650	570
25	500	530	550
50	900	930	950
75	900	1600	1840
100	1300	1250	1250
125	1900	1110	1160
150	1600	1760	1830
175	2100	2520	2500
200	2000	2350	2410
225	2500	2510	2530
250	2600	2510	2540

Table A2.2. Force measurements without foam

value	Run 1(g)	Run 2(g)	Run 3(g)
0	1930	3000	2930
25	2920	1500	3000
50	3000	2950	2900
75	2900	2320	3000
100	2400	2850	2830
125	2950	2880	3000
150	2600	2440	2900
175	2940	2920	3000
200	2850	2960	2850
225	3000	2900	2970
250	3000	2900	2930

Appendix 3. gripper registers

Table A3.1. Gripper registers

request

Byte name	Bit	name
Action request	0	rACT
	1 2	rMOD
	3	rGTO
	4	rATR
	5-7	rRS0
Gripper options	0	rGLV
	1	rAAC
	2	rICF
	3	rICS
	4-7	rRS1
Gripper options 2	0-7	rRS2
Position request A	0-7	rPRA
Speed A	0-7	rSPA
Force A	0-7	rFRA
Position request B	0-7	rPRB
Speed B	0-7	rSPB
Force B	0-7	rFRB
Position request B	0-7	rPRB
Speed B	0-7	rSPB
Force B	0-7	rFRB
Position request C	0-7	rPRC
Speed C	0-7	rSPC
Force C	0-7	rFRC
Pos req scissor	0-7	rPRS
Speed scissor	0-7	rSPS
Force scissor	0-7	rFRS

status

Byte name	Bit	name
Gripper status	0	gACT
	1 2	gMOD
	3	gGTO
	4 5	gIMC
	6 7	gSTA
Object status	0 1	gDTA
	2 3	gDTB
	4 5	gDTC
	6 7	gDTS
Fault status	0-3	gFLT
	4-7	gRS1
Pos. req. A echo	0-7	gPRA
Position A	0-7	gPOA
Current A	0-7	gCUA
Pos. req. B echo	0-7	gPRB
Position B	0-7	gPOB
Current B	0-7	gCUB
Pos. req. C echo	0-7	gPRC
Position C	0-7	gPOC
Current C	0-7	gCUC
Pos. req. scissor echo	0-7	gPRS
Position scissor	0-7	gPOS
Current scissor	0-7	gCUS