DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Report for Practical Course on Control of Modular Robots

Interfacing Simulink with the Robot using a Rapid Control Prototyping System

Authors:	Mohit Arunkumar Agarwal
Supervisor:	Andrea Giusti
Submission Date:	21.01.2016

Abstract

The author discusses about modular robots and how it can be controlled on a Rapid Control Prototyping System using Simulink on the workstation. The report starts by briefly describing the various components in use for the Praktikum, and the protocols that are important for making the communication possible. The objective of the Praktikum is to have a control design for one of the axes of the robot. However, because the development of the model was started from scratch, the author shows how the various building blocks of the model are developed in Simulink system. Using these, new blocks can be developed by just picking and placing them in the new model. The author then discusses about the safety considerations by reading ACK signal, position and velocity interpretation and then finally the design of a PI controller for the system.

Contents

1 Introduction

In this Praktikum, the author has interfaced a Modular Manipulator Robot arm with a Rapid Control Prototyping System using CAN bus protocol in MATLAB/Simulink. This helps to achieve real time prototyping and debugging of the Robot system, hence, in turn reducing maintenance time and repeated compiling and dumping of code on the system. Given below are brief descriptions of the major technologies involved in the praktikum.

1.1 Modular Robots

Reconfigurable and Modular Robots are mechatronic systems contained of interchangeable modules. As the name suggests, the individual mechanical components of the robot are controlled separately. This helps in achieving flexibility in operation, for example, such robots can be assembled in the form of a snake like structure to navigate through tiny spaces, or a spider-like structure to overcome difficult terrain.

Modular Robots is a field with immense promise and widespread applications. These robots can assume the shape according to the requirement. This obviates the need of specialized robots for each task. Hence, this brings down the operating and maintenance costs, as customers can just buy additional modules when required, or replace the damaged modules easily without compromising its functionalities.

1.2 Rapid Control Prototyping Systems (RCP Systems)

Rapid Control Prototyping (RCP) is a process that lets engineers quickly test and iterate their control strategies. Consequently, mathematical models are automatically imported with MATLAB/Simulink on a real-time machine with real I/O interfaces to connect to real-world systems. RCP systems help in automating much of the time-consuming work involved enabling the developers to:

- Test new system designs and algorithms in hardware in real-time
- Perform design operations in minutes
- Focus on innovation

Benefits – RCP decreases development time by allowing corrections to be made early in the product process. By giving engineering a look at the product early in the design process, mistakes can be corrected and changes can be made while they are still inexpensive.

1.3 CAN Protocol

The CAN bus protocol is defined by the ISO 11898-1 standard and can be summarized like this:

- The physical layer uses differential transmission on a twisted pair wire.
- A non-destructive bit-wise arbitration is used to control access to the bus.
- The messages are small (at most eight data bytes) and are protected by a checksum.
- There is no explicit address in the messages, instead, each message carries a numeric value which controls its priority on the bus, and may also serve as an identification of the contents of the message.
- An elaborate error handling scheme that results in retransmitted messages when they are not properly received.
- There are effective means for isolating faults and removing faulty nodes from the bus.

1.4 Schunk Motion Protocol

Schunk has described it’s protocol for sending commands to the modular robot using CAN bus, Profibus and RS232. The detailed description of the protocol and how to send the commands can be found in the Schunk Motion Manual.

2 Objectives

The main objective of the Praktikum is to achieve control of the modular manipulator using the RCP system in real time. The sequential checkpoints to reach the target are:

1. Design subsystems for the required CAN send commands as per the Schunk motion protocol
2. Get the state of the robot at regular intervals and decrypt the state to read position and velocity of each module
3. Compare integrated velocity state against the read position state
4. Establish the safety condition to stop the robot if ACKs are not received for several sent CAN messages
5. Design a PI controller for the system for smooth movements and enhance safety of movement
6. Test the model at different baudrates and find the most suitable rate for communication

3 Equipment

3.1 Schunk LWA 4P

The robotic arm integrates modular servo-actuators of 2 Degrees Of Freedom, called ERB. The LWA 4P module incorporates a power amplifier and a controller, so it does not require an external control cabinet. This robot model is in configuration of 6 Degrees Of Freedom. It is powered by 24V DC, hence it does not need an external large inverter.

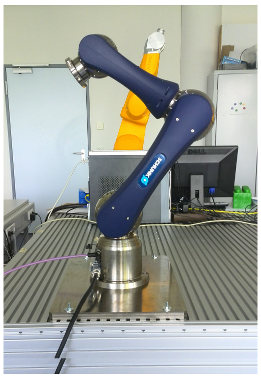

Figure 3.1: Schunk LWA 4P Manipulator arm

The technical specifications of the robot can be found in the data sheet.

3.2 Speedgoat Performance real-time Target Machine

The Speedgoat Performance real-time Target Machine is the heart of the Praktikum. It is an RCP system responsible for real-time debugging and prototyping of the manipulator arm. This machine has been specifically designed for laboratory use and research purposes by Speedgoat.

The technical specifications of the machine can be found in its datasheet.

Figure 3.2: Speedgoat rapid control prototyping system

3.3 MATLAB/Simulink

MATLAB/Simulink is a powerful tool for signal modelling of integrated systems. It boasts of a large blockset library to ease development and signal interfacing. It already includes CAN library which has the blocks for CAN Unpack and CAN Pack, thus, the user can just provide the data for the CAN message. Speedgoat also has developed blocksets for communication with the real-time target machine using Simulink. The author could focus on improving the performance of the system on rather than diving into specifics of communication protocols.

3.4 Setup

The entire apparatus for the Praktikum is as shown in Figure 3.3

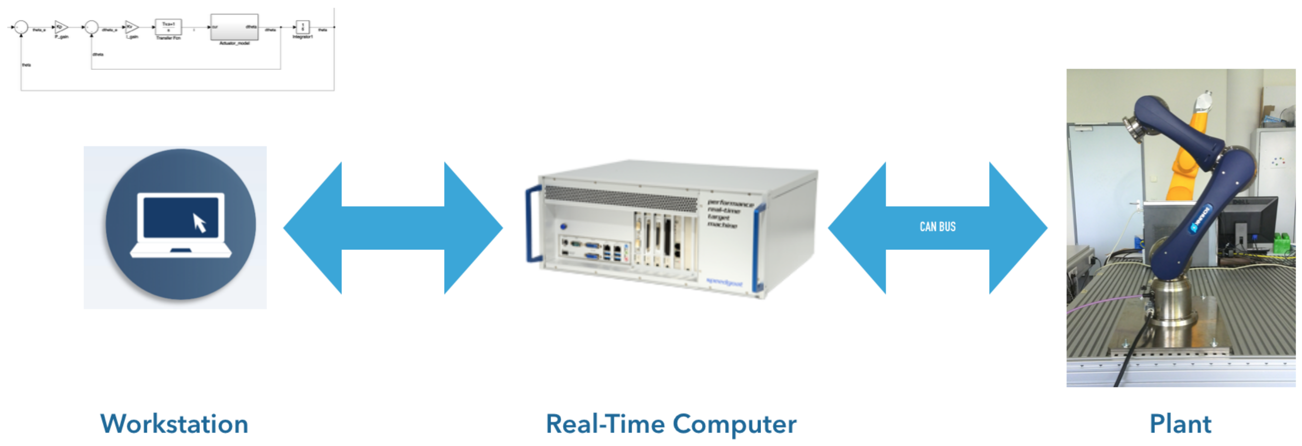

Figure 3.3: Equipment Setup

4 Observations

The robot's movement is controlled in the following manner:

1. At 1s, the emergency stop error is removed (QUIT ERROR command)
2. At 2s, the robot is brought to it's default configuration, i.e. all modules at position 0 millidegrees (MOV POS command)
3. At 10s, the robot is moved by sending MOV VEL command at each sample time with current as the parameter
4. At 14s onwards, the MOV VEL command is no longer sent
5. At 15s, the robot is commanded to return to it's starting position

4.1 CAN bus commands

The author developed the the following CAN commands with Schunk motion protocol in Simulink. The Outputs of all the Command blocksets is in accordance with the Schunk Motion Protocol. This was tested using PEAK-CAN-USB driver and verified against the sent messages in MTS tool from Schunk.

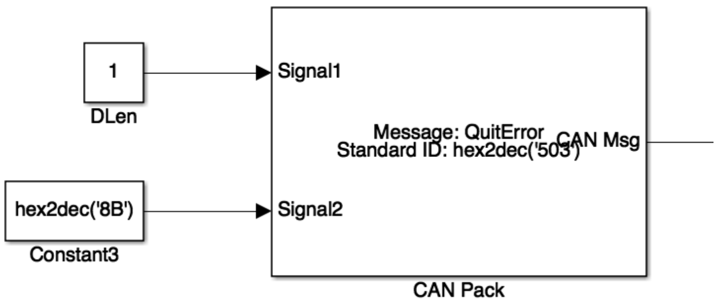

Figure 4.1: Quit Error

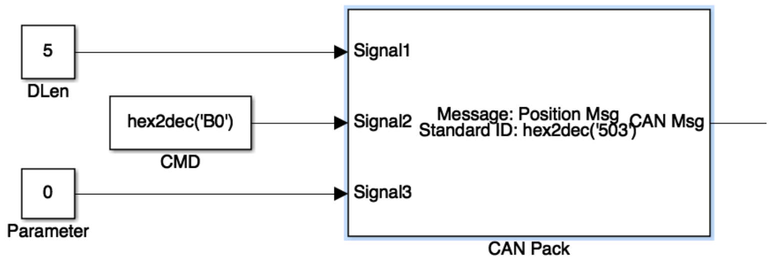

Figure 4.2: Move to Position

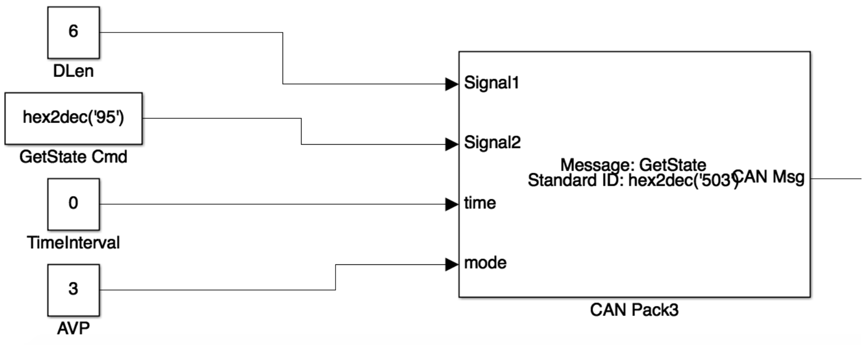

Figure 4.3: Get State Once

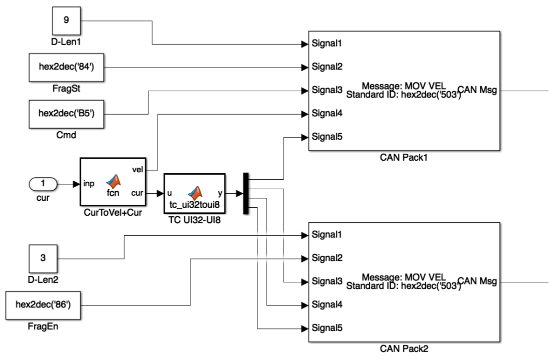

Figure 4.4: Move at velocity with Current Parameter

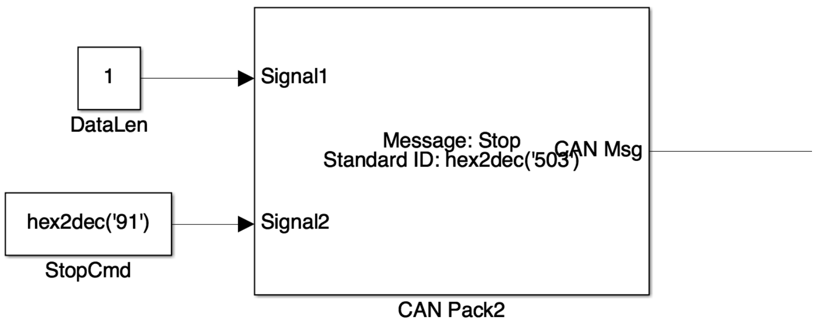

Figure 4.5: STOP command

The MTU of a CAN message is 8 bytes. However, the MOV VEL command in Figure 4.4 exceeds the 8 bytes. The Schunk motion protocol describes the use of Fragmentation in such cases. As a result, the final command has two CAN Pack messages, one for each fragment. The velocity is determined in the MATLAB function block. When the current has negative value, the velocity parameter is signed as negative and the current parameter is sent as positive. Similarly, with positive current value, the velocity parameter becomes positive while the current parameter remains positive.

The current and velocity parameters in Figure 4.4 are determined using the following MATLAB function:

```
function [vel, cur] = fcn(inp)

if inp > 0
    vel = int32(90000);
else
    vel = int32(-90000);
end

c1 = (abs(inp));
cur = uint32(c1);
end
```

4.2 Decrypting the State

The robot was sent a Get State command (Figure 4.3) at every sample time with the MOV VEL (Figure 4.4) with the Time Parameter set to zero. This resulted in the Robot sending it's status each time the it received a MOV VEL command.

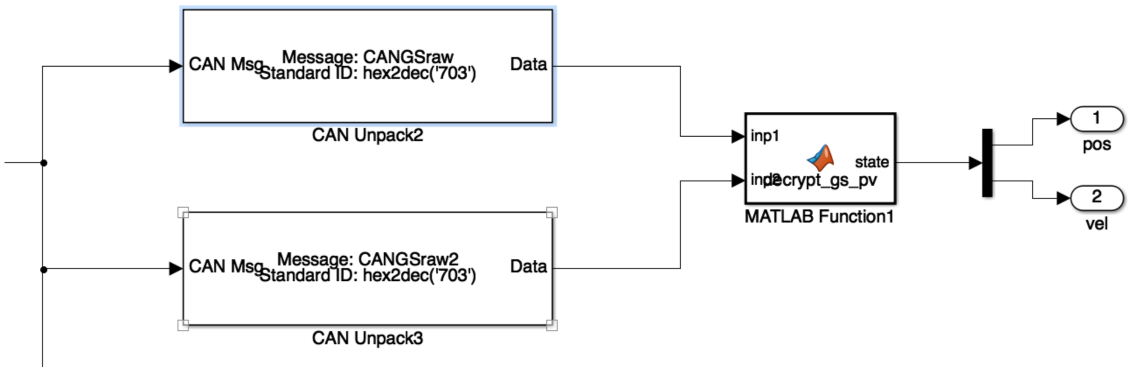

Figure 4.6: Reading Status of the Robot

The decryption block (Figure 4.6) consists of two CAN unpack block because the robot is sending it's position and velocity parameters through fragmentation according to Schunk Motion Protocol. Hence, it is necessary to combine the two messages and find out the corresponding position and velocity parameters. According to the Schunk Motion Protocol, the CAN message for the status of the robot is fragmented as per the following:

First CAN Message 1

1. Byte 1: Total length of the CAN message including the fragmented message
2. Byte 2: Fragment code (0x84 for first fragment)
3. Byte 3: Command Code (0x95 for Get State)
4. Byte 4-7: Position parameter (Data type `single`)
5. Byte 8: one out of four bytes of velocity parameter (Data type `single`)

Second CAN Message 2

1. Byte 1: number of remaining CAN bytes
2. Byte 2: Fragment code (0x86 for final fragment)
3. Byte 3-5: Remaining 3 bytes of velocity parameter (Data type `single`)

This is implemented by the MATLAB Function1 in the Figure 4.6. The code of the function is as follows:

```
function state    = decrypt_gs_pv(inp1, inp2)

persistent pos;         % Position value
persistent vel;          % Velocity value

if isempty(pos) || isempty(vel)
   pos = zeros(1,4);
   vel = zeros(1,4);
end

if inp1(3) == hex2dec('95') % Check if it is Get State response
    pos(1:4) = uint8(inp1(4:7));
    if inp1(1) > 8
        vel(1) = uint8(inp1(8));
        vel(2:4) = uint8(inp2(3:5));
    end
end
pos1 = typecast(uint8(pos), 'int32');
vel1 = typecast(uint8(vel), 'int32');
state = [pos1 vel1];    % Current state
end
```

4.3 Interpreting the State

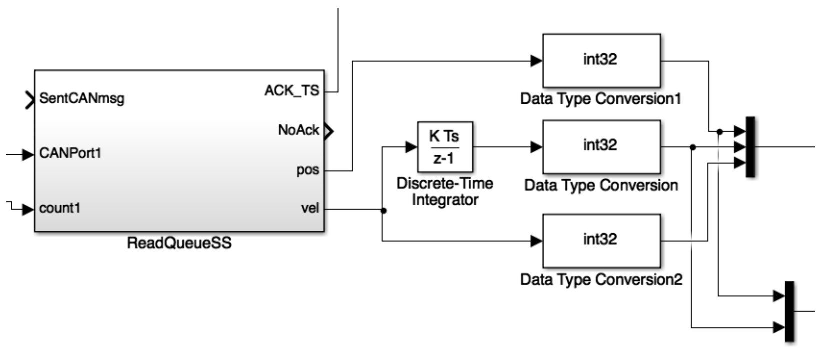

Figure 4.7: Read position and integrated velocity

To enhance communication speed, it is of interest to only determine either the position or the velocity of the robot so that the status signal is only one CAN message long. This will help in reducing the collisions on the CAN bus and thus boost the speed of communication.

However, it is crucial to determine both the position and velocity of the robot for Control purpose. This makes it important to derive either the velocity by differentiating the read position, or integrate the read velocity to determine position. Differentiation of position over time results in a very noisy measured velocity. One solution can be to build a velocity observer, which is beyond the scope of this Praktikum.

Therefore, for testing purposes in this Praktikum, the author has read both the position and the velocity from the robot, integrated the velocity over time to measure the position and finally superimposed the measured position graph with the read position graph to understand the significance of this method (Figure 4.7).

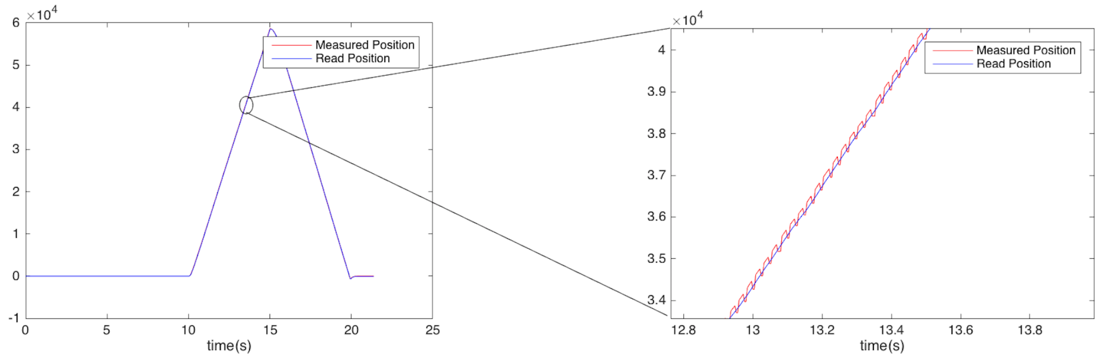

Figure 4.8: Comparing read position with integrated velocity

Figure 4.8 demonstrates that the integrated velocity measurement follow the read position closely. However on closer inspection it becomes clear that the integrated velocity is oscillating at an error of 50 millidegrees around the read position. This is significant error for the controller and thus, this method shall not be used to determine the position. A better solution will be to build a velocity observer.

4.4 Read ACKs

Every module, when sent a command, responds with a corresponding ACK command. This informs the controller that the module has received the sent command and is processing it. It is crucial to handle this command, the absence of it may mean that the module has malfunctioned. Hence, for the safety of the robot, if the controller fails to receive ACK for 5 subsequent sent CAN messages, it sends a STOP command (Figure 4.5) to the robot.

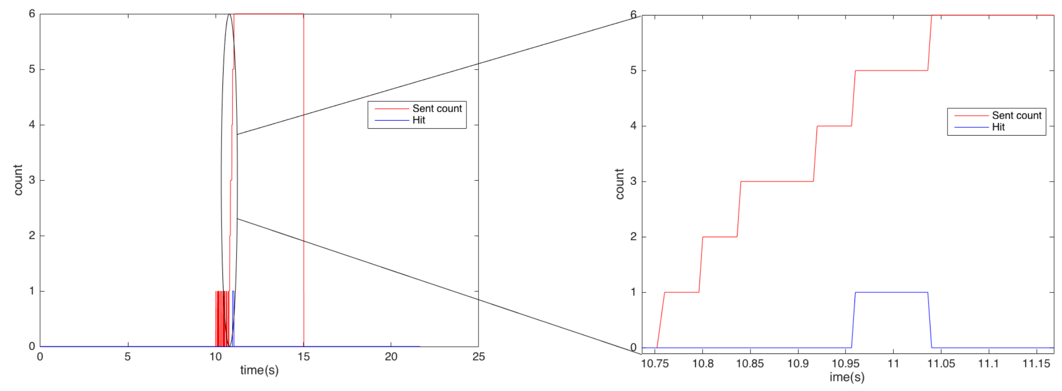

Figure 4.9: Counting Sent CAN messages and Sending Hit if ACKs not received

In Figure 4.9, it can be seen that when the model does not receive ACK for 5 subsequent messages, the counter is hit. The robot is now sent the STOP command (Figure 4.5) which increases the counter to 6. The count remains stable at 6 as no subsequent moving commands are being sent after the STOP command.

This is a crucial feature as safety of the robot movement is enhanced. It makes sense to stop the robot if there has been no response from the robot for several sent CAN messages; in this case, for 5 sent CAN messages.

To demonstrate this feature, the bus was deliberately filled with Get State commands (Figure 4.3). This resulted in a lot of collisions in the bus which deteriorated the receipt of ACK commands. The robot was asked to deliver it's state every 1 ms which further increased the rate of collisions. However, the robot is robust and it sends ACK signal every time it was sent a MOV VEL command (Figure 4.4), as demonstrated in Figure 4.10.

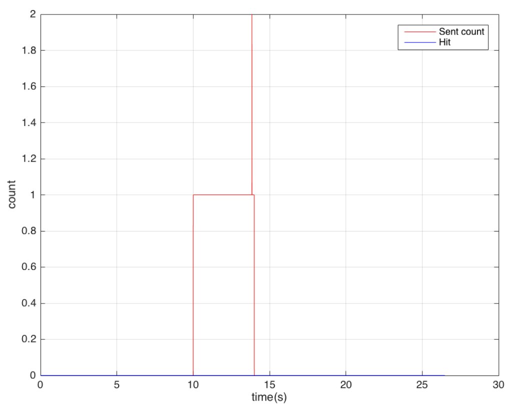

Figure 4.10: Counting Sent CAN messages with no ACKs

It can be interpreted easily that for every CAN message sent, the controller received the corresponding ACK as the count does not exceed 1 for most of the time. The maximum value the counter reaches in this run is 2, and therefore it's safe to say that the robot was working without any problems and there were not significant collisions on the bus.

4.5 Controller Design

An effective rejection of the disturbance torque (τ_d) on the output is ensured by:

- a large value of the amplifier gain before the point of intervention of the disturbance
- the presence of an integral action in the controller so as to remove the steady state error, if any

Hence, a PI controller is used in the forward path whose transfer function is:

$$C(s) = K_c \frac{1 + sT_c}{s} \tag{4.1}$$

It was ensured using position and velocity feedback that the general solution is preserved. Thus, to achieve the transfer function in Equation 4.1, the author used a cascading of blocks with position and velocity feedback.

The author first simulated the control system in Simulink with the Actuator model given as:

$$J_{eq}\ddot{\theta} = \tau_d - \beta_v\dot{\theta} + K_r K_m i \tag{4.2}$$

The corresponding Simulink model is as shown in figure 4.11

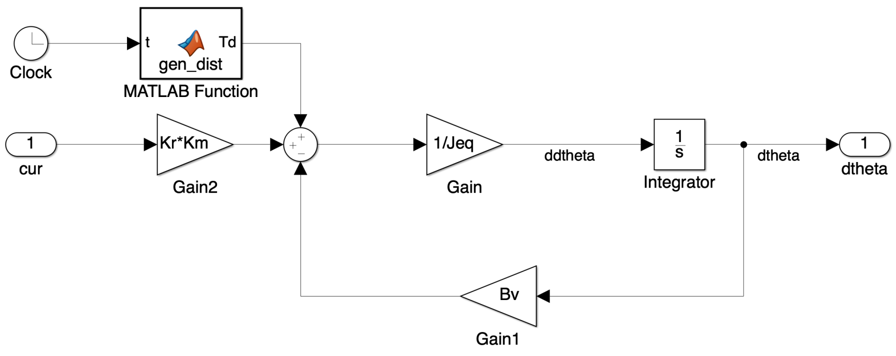

Figure 4.11: Simulink model for the Actuator

The controller parameters are chosen as:

$$C_p(s) = K_p \tag{4.3}$$

$$C_v(s) = K_v \frac{T_v s + 1}{s} \tag{4.4}$$

Hence, the control design of the actuator system looks like in Figure 4.12

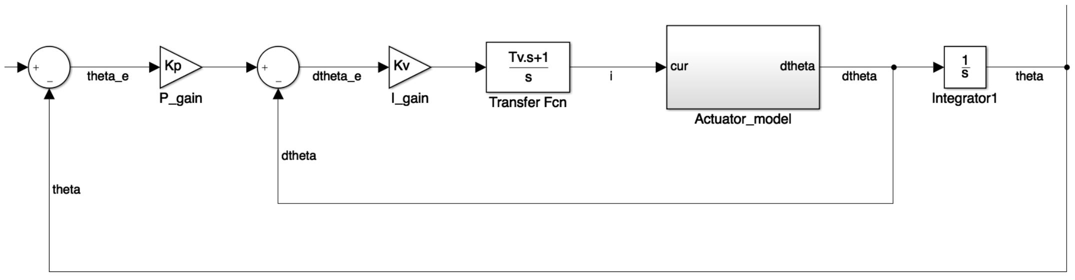

Figure 4.12: Simulink model PI Controller

The Transfer function of the whole system is:

$$\frac{\theta}{\theta_r} = \frac{1}{1 + s\frac{1}{K_p} + s^2 \frac{\beta_v}{K_p K_v K_r K_m}} \tag{4.5}$$

And the transfer function of a 2nd order system is given by:

$$\frac{Output}{Input} = \frac{1}{1 + s\frac{2\zeta}{w_n} + s^2 \frac{1}{\omega_n^2}} \tag{4.6}$$

4.5.1 Simulation results

Using critical damping (ζ=1) and ω_n equalling 10 times the sampling time (T_s), the response of the controller to a step input is as shown in figure 4.13

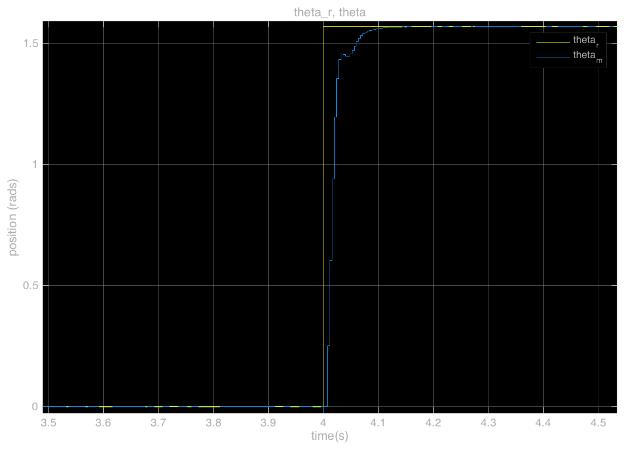

Figure 4.13: Output response to step input

It can be seen that the output response is closely following the step input, and smoothens out towards the peak. It's also worth noting that there are no overshoots. However, the control signal (i) becomes more than 600 A at the start of the step response as can be seen from the figure 4.14

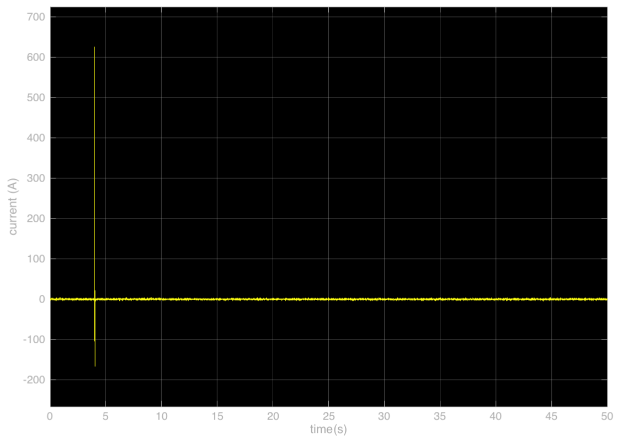

Figure 4.14: Control signal, i.e., Current value

This value of current is unreasonable. It can be seen that this high value of current is because of sudden jump in the signal value at $4s$. This can be prevented by having a smooth step signal. The smooth step signal is achieved by implementing the cubic polynomial:

$$x(t) = a_3t^3 + a_2t^2 + a_1t + a_0; \tag{4.7}$$

The cubic polynomial in Equation 4.7 is implemented by generating a MATLAB function that takes initial position(q_0), final position(q_n), start time(t_0), end time(t_n), initial velocity($\dot{q}_0$) and final velocity($\dot{q}_n$) as inputs. The PI controller output response is then seen in figure 4.15

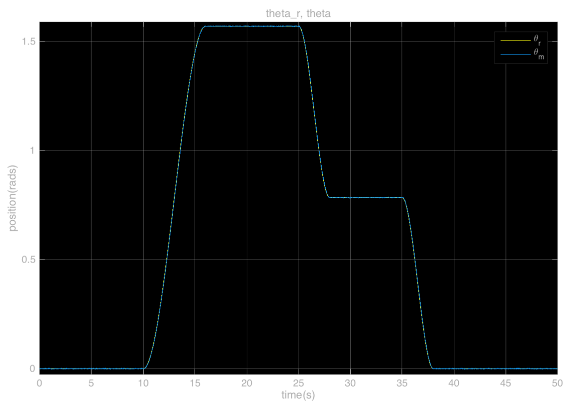

Figure 4.15: PI response with 3 smooth step signals as input

It is evident that the output signal almost overlaps with the input signal. The signal of interest, i.e. the control signal (i) is also within acceptable levels, as seen in figure 4.16

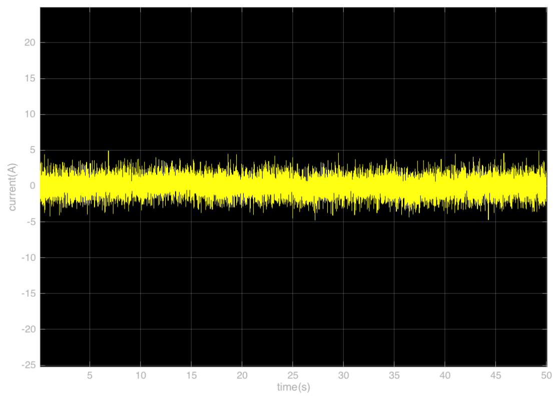

Figure 4.16: Current value with smooth step as input

4.5.2 Implementation on Robot

Using the same input signal as in figure 4.15, we read the response from the robot. It's crucial to understand the significance of ω_n in the real word system. A high value of ω_n can make the system unstable because of too fast a response time, whereas a low value may make the system too slow to respond to the input. By the rule of thumb, we calculate ω_n as:

$$\omega_n \geq 10 * T_s \tag{4.8}$$

This is the rule of thumb and the T_s for this system is 0.004s. Thus, $\omega_n \leq 157$.

Using ω_n = 150, $\zeta = 1$ and calculating the other parameters from Equations 4.5 and 4.6, the output of the robot is not very smooth as can be seen in figure 4.17

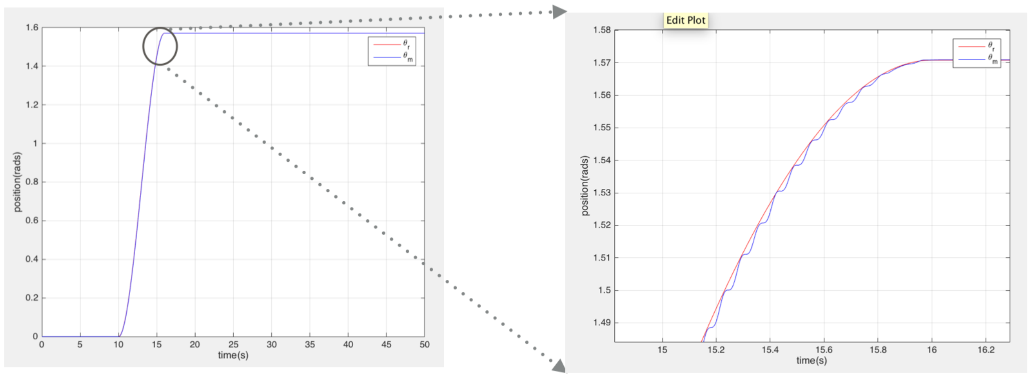

Figure 4.17: Response of robot at $\omega_n = 150$

Decreasing the ω_n to 120 yields a better response from the robot (Figure 4.18)

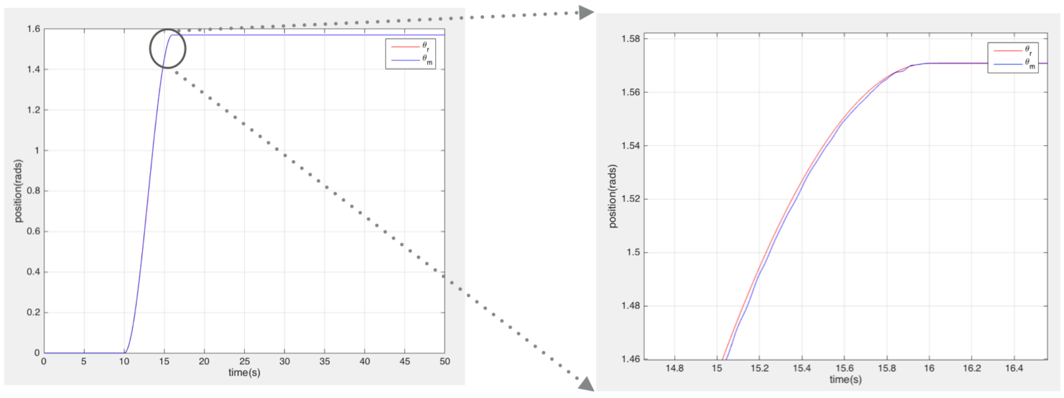

Figure 4.18: Response of robot at $\omega_n = 120$

5 Conclusion and Future Scope

5.1 Next Steps

First The author has so far developed all the systems to work on one axis (Axis ID 6). The immediate next step should be to try to make the system work on all the axes. The robot has 6 degrees of freedom and the Project cannot be deemed completed until all the axes can perform mutually. There can be several challenges: communication bandwidth may pose a bottleneck; independent joint control can cause control problems on other joints.

Second Simulink Real-time supports a utility called Simulink real-time explorer. It is possible to change parameters in real-time using the utility, thus, taking advantage of the real time control systems. Additionally, it is possible to develop a GUI in the Explorer. Using the GUI the student can read the status of the robot, display and distinguish the errors that could be encountered, change the parameters of the control system, as well as change the input signal to different position values.

Third The author has developed the system using CANBus Protocol. It is of interest to explore the possibility of using CANOpen for the same. CANBus protocol is a physical layer protocol whereas CANOpen is a higher layer protocol. Using the SDOs and PDOs, the repeated sending of signals to the robot can be avoided which leaves the CAN bus relatively free. This can be used to boost communication speed between the RCPS and the robot.

Fourth Currently, the robot has only one safety block, that is when it does not send an ACK signal, the STOP command is sent. However, it is crucial to implement a a more exhaustive safe block in which the system can check for unreasonable values of the parameters. For example, a check to ensure the Controller parameters do not make ω_n exceed $10T_s$, or a saturation block to avoid the current exceed 2000 mA.

5.2 Conclusion

It is evident that the control of the modular robot using RCPS is faster and efficient. The ability to change parameters in real-time using the Simulink real-time controller and to send new position commands without having to build the model significantly reduces the prototyping cost of the robot. The model is currently for only one axis and it was possible to design and test a control design for that axis in the matter of few days.

It is also interesting to observe that the robot is quite robust in itself, it sent an ACK signal for almost every CAN message sent to it. There were no emergency stop command sent to the robot during the control design period.

Additionally, it is also important to note that integrating velocity is not the solution. Thus, to have both the velocity and position of the robot, it is necessary to build a observer to obtain a better estimation of the velocity.

5.3 Future scope

The potential of modular robots is fascinating. Not only can these robots decrease the maintenance costs, but manufacturing of modules will be a lot cheaper and faster. It is potentially possible that these robots be autonomous and later be self-reconfigurable, thus removing human action from the loop. This will further instil a plug-and-play architecture for the robot, in which the new modules is simply plugged in and the robot is ready to act without manually having to make changes to the parameters and it's mathematical model.

List of Figures