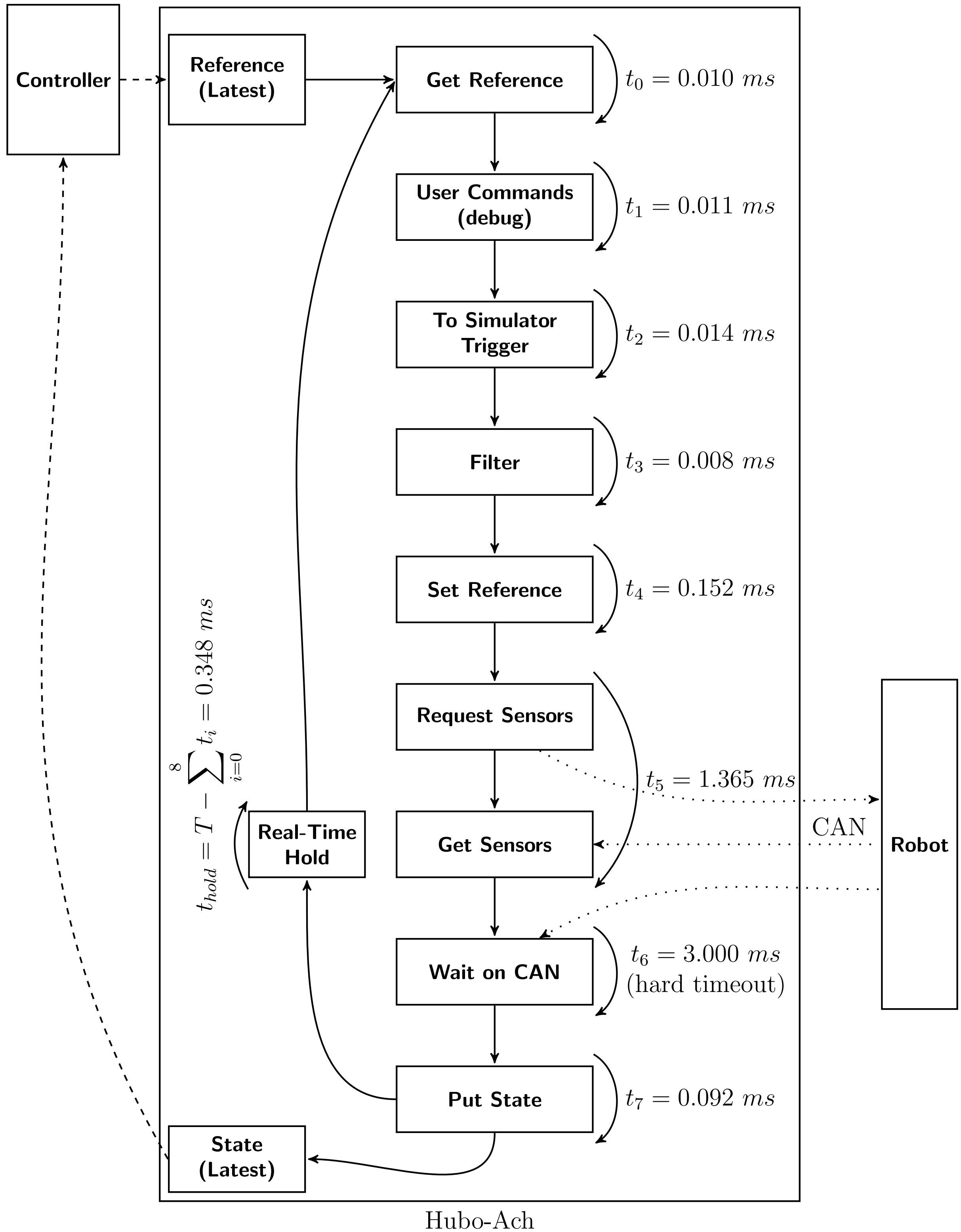
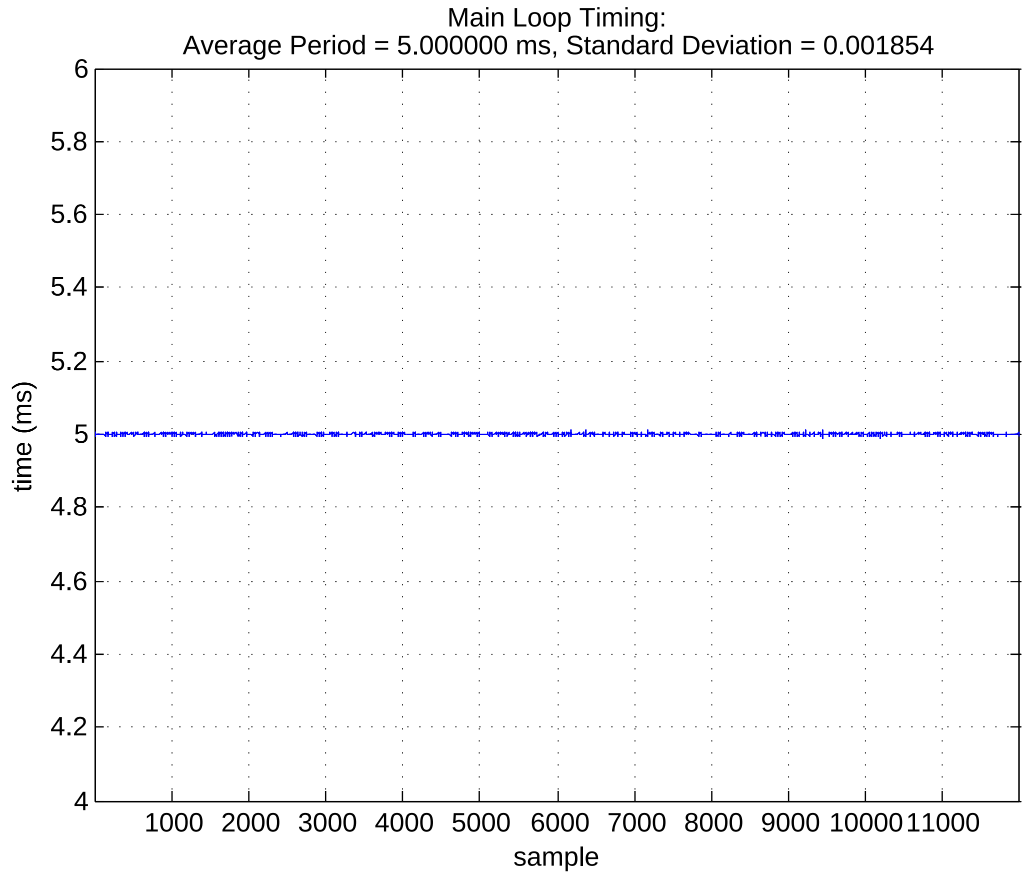


Figure 4.3: Timing diagram of Hubo-Ach. All times t_* denote measured times each block takes to complete. Tests were done on a 1.6Ghz Atom D525 Dual Core with 1GB DDR3 800Mhz memory running Ubuntu 12.04 LTS linux kernel 3.2.0-29 on a Hubo2+ utilizing a CAN bus running at 1Mbps baud. Average CPU usage is 7.6% using a total of 4Mb or memory.



Main Loop Timing:
Average Period = 5.000000 ms, Standard Deviation = 0.001854
time (ms)
6
5.8
5.6
5.4
5.2
5
4.8
4.6
4.4
4.2
4
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
11000
sample

Table 4.2: Inter Process Comunication Method Comparison

Inter-Process Comunication Method	Open Source	POSIX Complaint	Non Blocking	Multiple Senders and Receivers	Low Latency	Light Weight	Access Old Data
Streams	yes	yes	no	yes	no	yes	yes
Datagram Sockets	yes	yes	no	yes	no	yes	yes
POSIX Message Queues	yes	yes	no	yes	no	yes	yes
Shared Memory	yes	yes	yes	yes	yes	yes	no
AIO	yes	yes	yes	yes	yes	yes	yes
CORBA	yes	yes	yes	no	yes	yes	yes
ROS	yes	yes	no	yes	no	no	no
Data Distribution Service	yes	yes	yes	yes	yes	yes	yes
Ach	yes	yes	yes	yes	yes	yes	yes

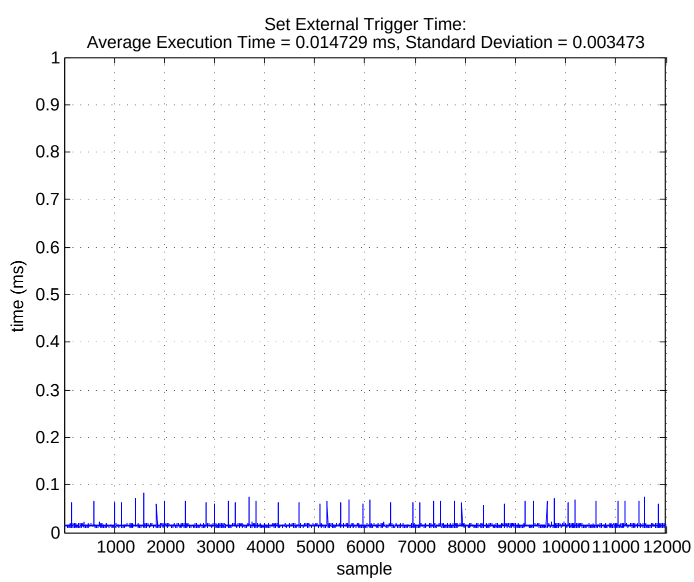


Figure 4.4: The amount of time it takes to send the external trigger. In this case each sample has a time step of 0.005 *sec*

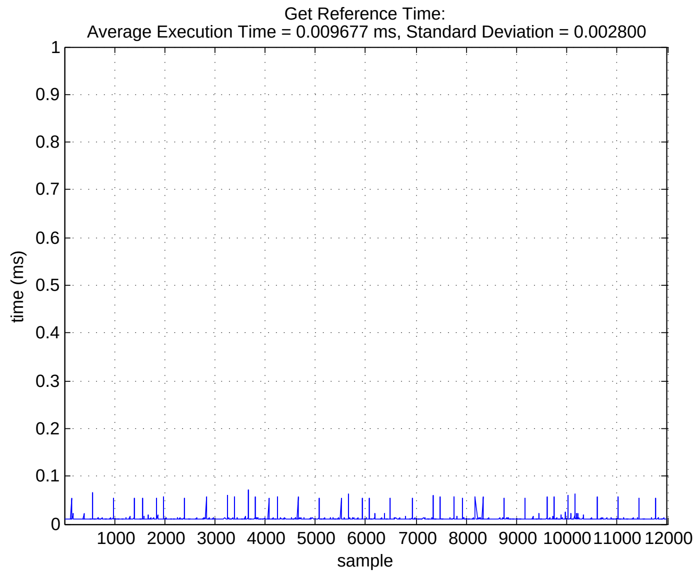


Figure 4.5: The amount of time it takes to request and get the reference for the actuators. In this case each sample has a time step of 0.005 *sec*

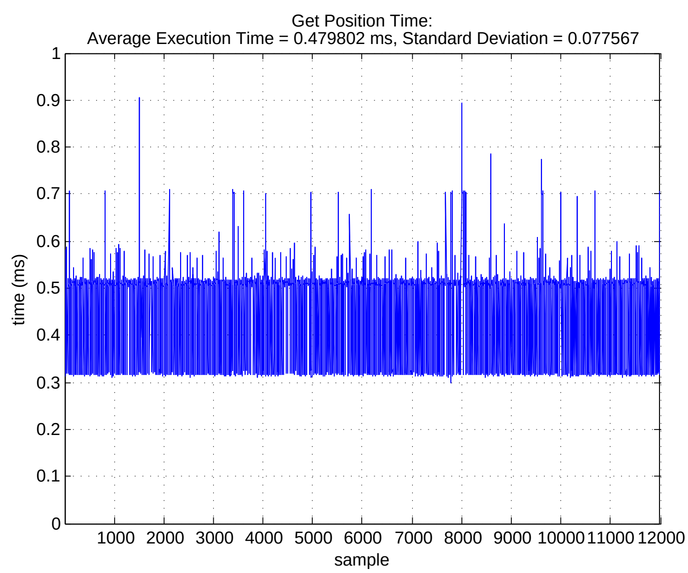


Figure 4.6: The amount of time it takes to request and get the actual position from the actuators. In this case each sample has a time step of 0.005 *sec*

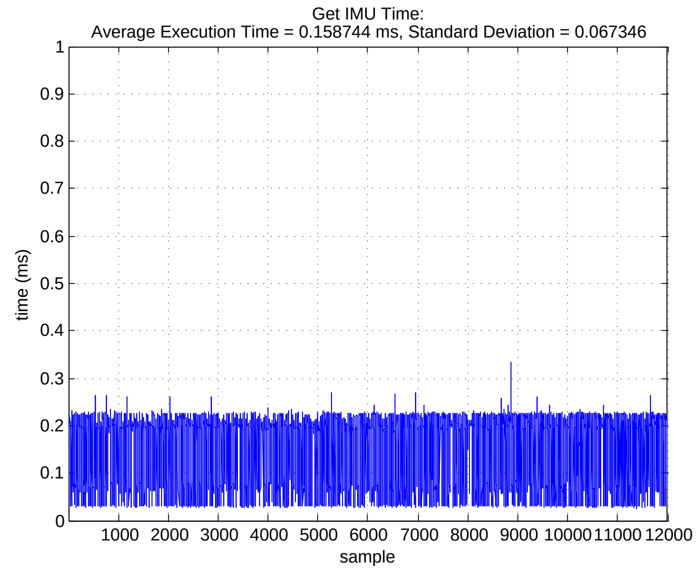


Figure 4.7: The amount of time it takes to request and get the IMU data. In this case each sample has a time step of 0.005 *sec*

Figure 4.8: The amount of time it takes to request and get the accelerometers data. In this case each sample has a time step of 0.005 *sec*

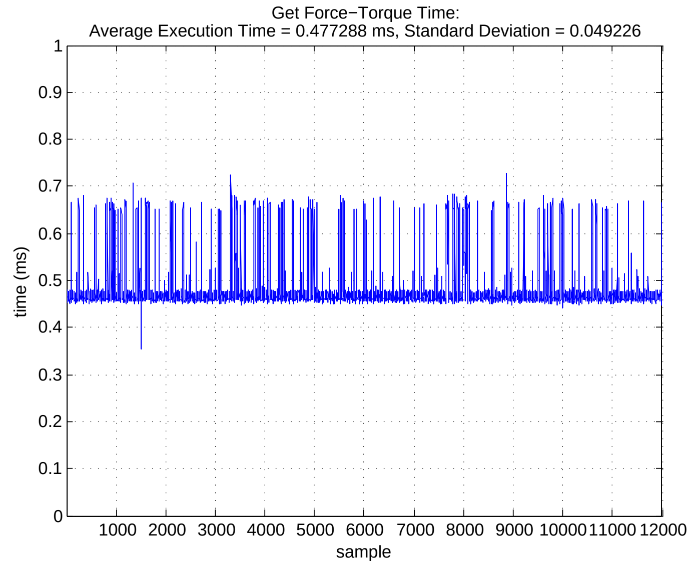


Figure 4.9: The amount of time it takes to request and get the force-torque sensors. In this case each sample has a time step of 0.005 *sec*

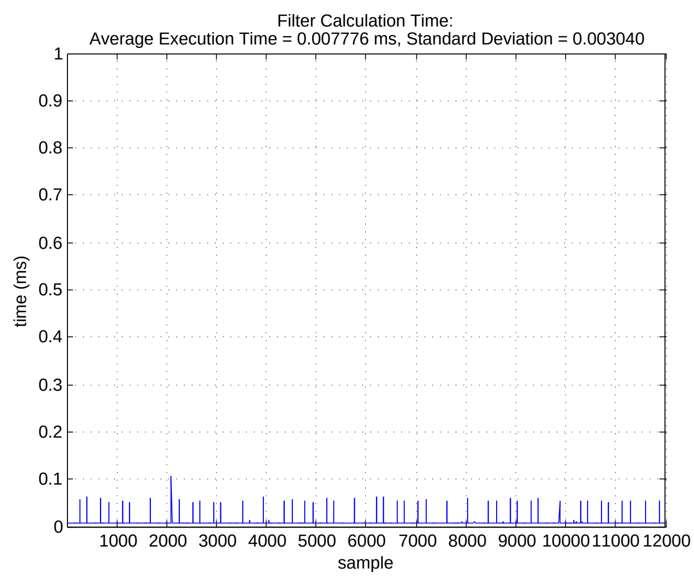


Figure 4.10: The amount of time it takes to process the built in filter. In this case each sample has a time step of 0.005 *sec*

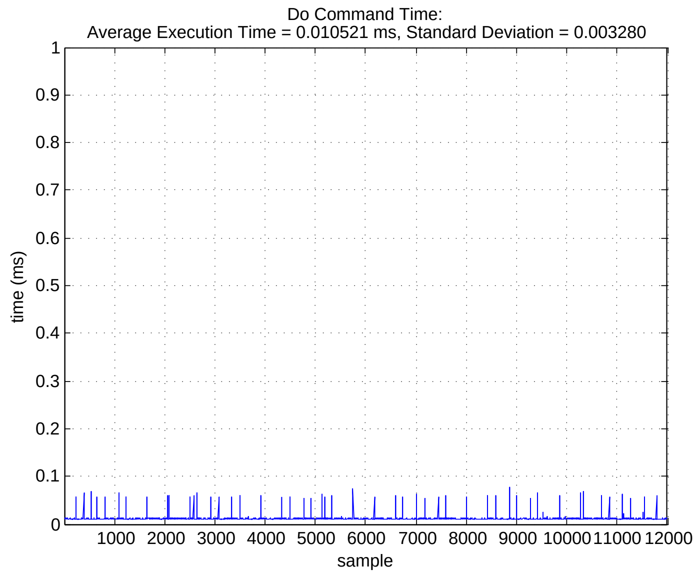


Figure 4.11: User command timing per sample. In this case each sample has a time step of 0.005 *sec*

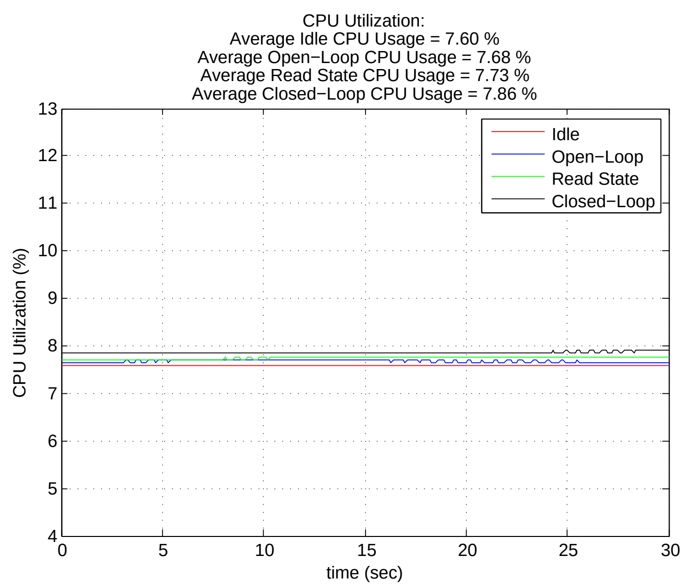
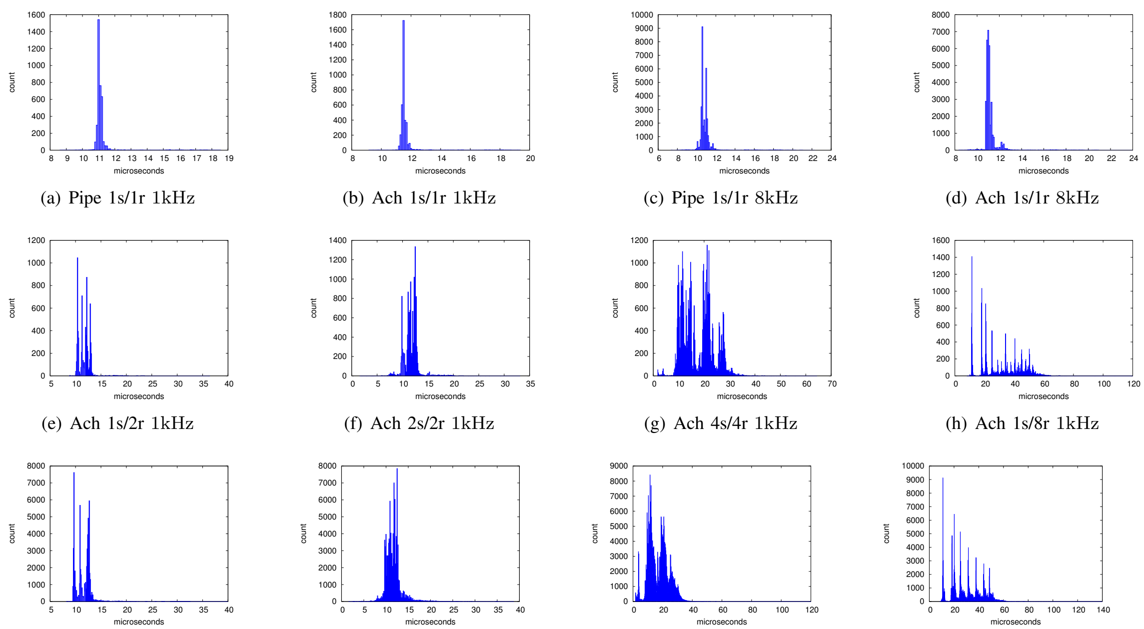


Figure 4.12: CPU utilization for the Hubo-Ach process when 1) idle, 2) under open-loop control, 3) reading the sensors, and 4) under closed-loop control. It is important to note that the cpu utilization stays within 0.3% when idle and under closed loop control. This means that the CPU utilization of Hubo-Ach is independent of the external control method. Thus it will not add more to the CPU load under complex control schemes then under simple ones.

memory there is no way of recovering older data that might have been missed by a controller.

What is needed is a method of sharing data that is *non-blocking* and as *low-latancy* like shared memory, but still holds older data and uses an asyncronous IO scheme. The asyncronous IO scheme is required so the controller is not locked to a set rate by the data transactionn method. N. Dantam et. al.[19] shows that Asynchronous IO (AIO) might be approperiate for this application however the implimentaiton under Linux is not as mature as I require. In addition N. Dantam shows that other IPC mechanism using select/poll/epoll/kqueue are widely used network server and help midigate but not totally removed the issue of HOL. The primary problem being that that thought the sender will not block the reader must stil read the oldest data first. The question now is what IPC mechanism will be suitable for my control system.



(a) Pipe 1s/1r 1kHz (b) Ach 1s/1r 1kHz (c) Pipe 1s/1r 8kHz (d) Ach 1s/1r 8kHz

(e) Ach 1s/2r 1kHz (f) Ach 2s/2r 1kHz (g) Ach 4s/4r 1kHz (h) Ach 1s/8r 1kHz

(i) Ach 1s/2r 8kHz (j) Ach 2s/2r 8kHz (k) Ach 4s/4r 8kHz (l) Ach 1s/8r 8kHz

Figure 4.2: Histograms of Ach and Pipe messaging latencies. Benchmarking performed on a Core 2 Duo running Ubuntu Linux 10.04 with PREEMPT kernel. The labels αs/βr indicate a test run with α sending processes and β receiving processes[19].