# Timing analysis of the motor control code

## Code setup

Execution times of functions below were obtained using the following code template:

|  |
| --- |
| #include <chrono>  ...  auto start = std::chrono::high\_resolution\_clock::now();  <*function* to be checked for execution time>;  auto end = std::chrono::high\_resolution\_clock::now();  auto duration = std::chrono::duration\_cast<std::chrono::microseconds>(end - start);  std::cout << "Execution time of *function*: " << duration.count() << " microseconds" << std::endl; |

## Investigated code

|  |
| --- |
| //function to loop through all MotorGroup objects to calculate new value of torque  void ActuationSystem::update\_torque(){  std::vector<std::thread> threads; //vector to store threads  //loop to put torque update function threads into the vector  for (int i = 0; i < \_number\_of\_converters; i++) {  threads.emplace\_back(std::thread(&MotorGroup::update\_torque, \_motor\_groups[i]));  }  //loop to merge the threads  for (auto& t : threads) {  t.join();  }  } |
| update\_torque() function in ActuationSystem class iterating through all connected converters. |

|  |
| --- |
| void MotorGroup::update\_torque(){  //a loop to iterate through all the motors  //std::cout << "Entered MotorGroup::update\_torque()"<< std::endl;  for(int i = 0; i < \_number\_of\_motors; i++){  //auto start = std::chrono::high\_resolution\_clock::now();  sendRecv(&\_motors[i].motor\_run, &\_motors[i].motor\_r); //send an initial message to the motor and receive data from it  extract\_data(&\_motors[i].motor\_r); //extract data from the received message  //auto end = std::chrono::high\_resolution\_clock::now();  //auto duration = std::chrono::duration\_cast<std::chrono::microseconds>(end - start);  //std::cout << "Execution time of sendRecv() and extract\_data(): " << duration.count() << " microseconds" << std::endl;  \_motors[i].new\_torque(); //update motor's torque  }  } |
| update\_torque() function in MotorGroup class iterating through all connected to a converter motors. |

## Test setups

**Setup 1:**

* USB hub connected
* 2 converters connected
* 1 motor on converter 0 and 3 motors on converter 1
* Update\_torque() function in ActuationSystem has **no** thread function paralellising the for loop

Result:

|  |  |
| --- | --- |
| Function | Measured execution time [μs] |
| ActuationSystem::Read\_actual\_positions() | 1 |
| ActuationSystem:: Send\_desired\_positions | 8 |
| ActuationSystem:: Update\_torque | 4000 |

**Setup 2:**

* USB hub connected
* 2 converters connected
* 1 motor on converter 0 and 3 motors on converter 1
* Update\_torque() function in ActuationSystem has thread function paralellising the for loop

Result:

|  |  |
| --- | --- |
| Function | Measured execution time [μs] |
| ActuationSystem::Read\_actual\_positions() | 1 |
| ActuationSystem:: Send\_desired\_positions | 8 |
| ActuationSystem:: Update\_torque | 3000 |

**Setup 3:**

* **No** USB hub connected
* 1 converter
* 3 motors on converter 0
* Update\_torque() function in ActuationSystem has thread function paralellising the for loop

Result:

|  |  |
| --- | --- |
| Function | Measured execution time [μs] |
| ActuationSystem::Read\_actual\_positions() | 1 |
| ActuationSystem:: Send\_desired\_positions | 7 |
| ActuationSystem:: Update\_torque | 3000 |

**Setup 4:**

* **No** USB hub connected
* 1 converter
* 3 motors on converter 0
* Update\_torque in ActuationSystem has **no** thread function paralellising the for loop

Result:

|  |  |
| --- | --- |
| Function | Measured execution time [μs] |
| ActuationSystem::Read\_actual\_positions() | 1 |
| ActuationSystem:: Send\_desired\_positions | 8 |
| ActuationSystem:: Update\_torque | 3000 |

**Setup 5:**

* **No** USB hub connected
* 1 converter
* 3 motors on converter 0
* Update\_torque in ActuationSystem has **no** thread function paralellising the for loop
* Added one time measurement directly for both sendRecv and extract\_data functions in MotorGroup::update\_torque().

Result:

|  |  |
| --- | --- |
| Function | Measured execution time [μs] |
| ActuationSystem::Read\_actual\_positions() | 1 |
| ActuationSystem:: Send\_desired\_positions | 7 |
| ActuationSystem:: Update\_torque | 3000 |
| SerialPort::sendRecv() + SerialPort::extract\_data() | 1000 |

**Setup 6:**

* **No** USB hub connected
* 1 converter
* 3 motors on converter 0
* Update\_torque() in ActuationSystem hasthread function paralellising the for loop
* sendRecv and extract\_data functions are parallelised in MotorGroup

Result:

|  |
| --- |
| Entered the second loop  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  [WARNING] SerialPort::recv, unblock version, wait time out  [WARNING] motor id=0 does not reply  [WARNING] Receive data CRC error  [WARNING] SerialPort::recv, unblock version, wait time out  [WARNING] motor id=1 does not reply  [WARNING] Receive data CRC error  [WARNING] SerialPort::recv, unblock version, wait time out  [WARNING] motor id=2 does not reply  [WARNING] Receive data CRC error  Execution time of update\_torque(): 21956 microseconds. Max f: 45.5456 |
| Terminal output |

## Conclusion

SerialPort:: sendRecv() and SerialPort:: extract\_data() functions are a bottleneck for a quick operation of the software as they take 1 ms every time they are called, which for 3 motors per converter amounts to total execution time of 3 ms for ActuationSystem::update\_torque(). This limits the program to max frequency at which it can be run of 330 Hz. The required frequency is 500 Hz.