Couch + KIM Emulator

// Investigate system latency impact on couch compensation performance

// Seperating function into two threads

// 1. Tracking thread

// This thread reads motion input trace, add noise, send out KIM measurement at every 350ms. This thread should also receive information from couch compensation thread so that the new positions being sent out are updated.

// 2. Couch compensation thread

// This tread reads measurements from tracking thread and simulates the couch response. The algorithm of how couch moves and only act on latest coming in data should be identical as the C++ code. Once couch starts moving,

// KIM latency: Wait for kV image being acquired, calculate marker position, send UDP signal

// Reads original motion trace

Input\_motion = read txt file(“file directory/filename”) // Trace should come from real patient motion trace (lung, liver, prostate etc.)

////////////////// KIM thread /////////////////////////

// Set up KIM emulator parameters

KIM\_latency = 700 // Variable that can be changed. Unit is in millisecond.

KIM\_sendFrequency = 350 // Variable that can be changed. Unit is in millisecond. The time interval of each measurement is sent to couch

KIM\_noise = 0.1 // Tracking target measurement generated by KIM has noise. Set a parameter to adjust noise level

couchFeedback = [(t1,a), (t2,b), (t3,c) …] // A list of timestamp and the corresponding position in one couch step. The total time duration should be a few hundreds ms.

Motion\_realtime = input\_motion + KIM\_couchFeedback // interpolation is needed. This trace represents the actual motion in real-world.

KIM\_read = KIM\_sendFrequency (interpolation) Motion\_realtime // KIM sends signal every 350ms

KIM\_send = KIM\_read x KIM\_noise // Taking into account of noise. This variable should store the data that is sent directly to couch via UDP.

System sleep(KIM\_latency) // Manually add system latency. Between tracking and signal being sent to couch, a latency exists.

KIM\_UDPSend //KIM\_send value to couch compensation thread

////////////////////// Couch Thread /////////////////////////////

// Set up couch response

// Couch receives measurement from KIM, if the incoming data is deviated from isocenter, it moves the same displacement but in opposite direction for compensation. The algorithm of when, how couch should move can be found in Ann’s code

// The target actual position will be updated due to couch. The new positions after compensation needs to be calculated and fed back to the KIM measurements.

Couch\_speed = 12 // Couch speed maximum 12mm/s. Load will decrease the motor speed.

Couch\_latency = 50 // latency between couch receives data, to initiate motion. Unit in milli-second.

Couch\_motionRange = 3 // Couch motion for compensation. Unit in milli-meter. The distance can be positive or negative.

Couch\_moveDuration = Couch\_motionRange/ Couch\_speed // Unit is in milli-second.

// Interpolation of couch position based on KIM input

// The couch position index matches with input trace, therefore the compensated position can be easily calculated.

(Couch\_time, couch\_position) = interpolation(KIM\_input\_time, KIM\_input\_position)

///////////////// Result demonstration //////////////////

Plot input trace

Plot KIM\_UDPSend

Plot (Couch\_time, couch\_position)