Tobii API 2

Thursday, September 14, 2006
9:22 AM

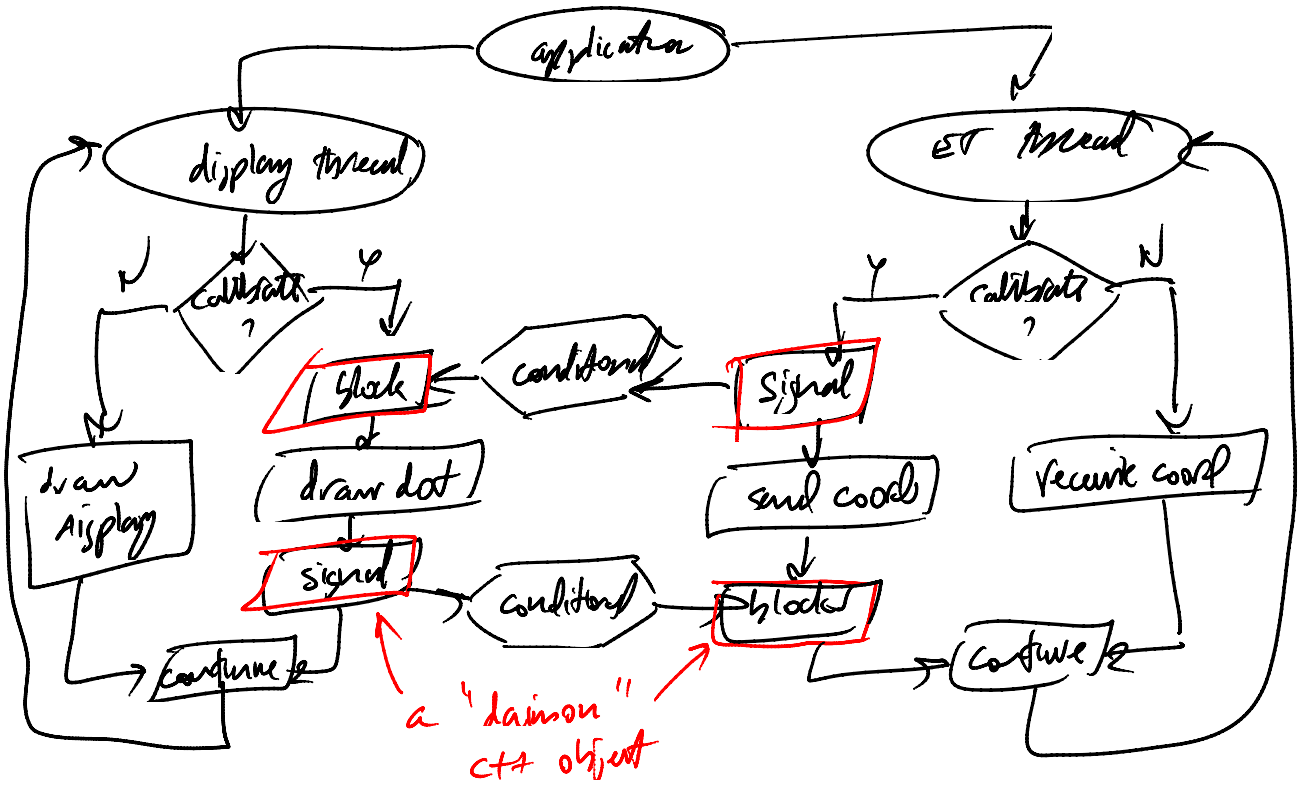
```
tobii_connect()
{
    Tet_Connect(state->tobii tHost, state->tobiiPort,
                                        "logfile")

    pthread_mutex_lock(&state->mutex)
    state->status.connected = 1;
    state->status.connect = 0;
    pthread_mutex_unlock(&state->mutex)
}
```

```
gazeDataReceiver(ETet_CallbackReason reason,
                 void *pData,
                 void *pApplicationData)

    SEt_GazeData *pGazeData = NULL;
    pGazeData = (SEt_GazeData *)pData;
    if (state->status.connected) {
        switch (reason) {
            case ET_CALLBACK_GAZE_DATA:
                if (state->status.running) {
                    pthread_mutex_lock(&state->mutex);
                    state->x = pGazeData->x_gazepos_lefteye;
                    state->y = pGazeData->y_gazepos_lefteye;
                    ...

if (state->
    status.running)
  && state->status.stop) {
    Tet_Stop();
    state->status.running = 0;
    ...
}
```

Thursday, September 14, 2006
9:22 AM

synchronization:

- important for calibration, especially
  & dynamic stimuli as well
- & use conditional vars (POSIX threads)
  ↳ prevents GUI/ET threads race condition
  ↳ it works, but ...
    - I'm not 100% sure it's the best solution
    - it may be slightly too complicated —
      simple boolean flags may suffice


application
display thread
ET thread
calibrate ?
Y
N
calibrate ?
Y
N
block
conditional
signal
draw display
draw dot
signal
conditional
send coords
block
receive coord
continue
continue
a "daimon" C++ object

```
tobii_calibrate()
{
  ...
  if (state->status.calibStarted) {
      // signal CUDA thread
      daimon->signal_gui(KEY_WRITE, sync, x, y),
      Tet_CalibAddPoint(x,y, sampler, gazeDataReceiver,
                        NULL, 0),
      if (sync) daimon->block_et();
      // advance to next calib pt
      ptr ++
```

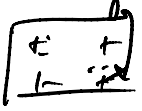
based on Butenhof's * client/server model

in my old code

daimon was called "server"

* Programming with POSIX Threads book.

```
// advance to next calib point
pthread_mutex_lock( & state -> mutex)
state -> calpoint++
pthread_mutex_unlock(---)
if ( last calib point) {
    // signal GUI to exit calibration
    daimon -> signal_gui( KEY_QUIT, false, -1.0, -1.0)
    ...
    Test_Calib CalculateAndSet()
    // should perform gravity check
                                    } calculate avg error
```



```
main_loop()
{
  if (state->status.connected) {
      if(state->status.calibrating &&
         state->status.calibstarted ) {
         // block until ET thread advances to next point
         if( daimon->block_gui(&x, &y)) {
               // draw calib dot at (x,y)
         }
         // signal ET thread to advance to next point
         daimon->signal_et();
      } else if (state->status.running) {
         // DO STUFF - e.g. draw gaze point at
                       (state->x, state->y)
                  or record gaze point, etc.

      }
}
```

Thursday, September 14, 2006
9:22 AM

Daimon interface:

```
typedef enum { REQ_READ, REQ_WRITE, REQ_QUIT } operation_t;
class Request {
    public:
    Request(operation_t op, float ix, float iy) : \
        operation(op), x(ix), y(iy) { }
    class Daimon;                    // friends
    operation_t  operation;
    float        x,y;
};
class Daimon {
    public:
    Daimon() : \
        synchronous(false), done_flag(false), request_flag(false)
        { pthread_mutex_init(&mutex, NULL);
          pthread_cond_init(&request, NULL);
          pthread_cond_init(&done, NULL);
        };
    void signal_gui(operation_t op, bool sync, float x, float y)
    bool block_gui(float dx, float dy),
    void signal_set(void);
    void block_set(void);
    private:
    deque<Request*>  queue;     // STL deque
    bool             synchronous;
    bool             done_flag    } predicates for
    bool             request_flag   wait
    pthread_cond_t   request
    pthread_cond_t   done
    pthread_mutex_t  mutex;
```

```
pthread_mutex_t mutex;
}
```

```
Daemon::signal_gui ( op, sync, x, y)
{
    pthread_mutex_lock (&mutex);
    synchronous = sync;
    // add new request to queue
    queue.push_back(new Request(operation, x, y))
    request_flag = true;
    // tell daemon a request is available
    pthread_cond_signal(&request);
    pthread_mutex_unlock (&mutex)
}
```

```
Daemon::block_gui(float &x, float &y)
{
    pthread_mutex_lock(&mutex)
    // wait for death
    while(!request_flag) pthread_cond_wait(&request
                                           &mutex);
    // it's possible that we missed signal of
    // request_flag was true to begin — check queue
    // before attempts to process death
    if(request_flag && !queue.empty()) {
        req = queue.front(); queue.clear();
        switch(req->operation){
            case REQ_QUIT:  break;
            case REQ_READ:  break;
            case REQ_WRITE: &x = req->x;
                            &y = req->y;
                            gotdata = true;
        }
        if(req) delete req;
    }
    request_flag = false
    return(gotdata);
}   // NOTE: mutex still locked
```

```
Daimon:: signal_et (void)
{
    if ( synchronous ) {
        done_flag = true;
        pthread_cond_signal (&done);
    }
    pthread_mutex_unlock(& mutex) ——— //see block_sig
}
```

(these two actions should be read as one continuous action)

```
Daemon :: block_et (void)
{
  pthread_mutex_lock (& mutex)
  while (! done_flag) pthread_cond_wait( & done,
                                         & mutex)
  done_flag = false;
  pthread_mutex_unlock( & mutex);
}
```