Kalman Filter

Real-time project

University of Pisa
Robotics and Automation Engineering

Gentili Alessandro
ID: 590268
mail: alex.gentili90@gmail.com

Venturini Pietro
ID: 516615
mail: pietroventurini95@gmail.com

7 November 2019

Abstract

This project aims at simulating a Kalman Filter that estimates and, if requested, predicts the current and future states of a particular system. The authors model two different systems: a user-mouse interaction and a mechanical system composed by a mass, which a randomly created force acts on. The implementation of the program uses allegro4.4 and *pthread* libraries beside an original library created to perform operations between matrices.

Kalman filter

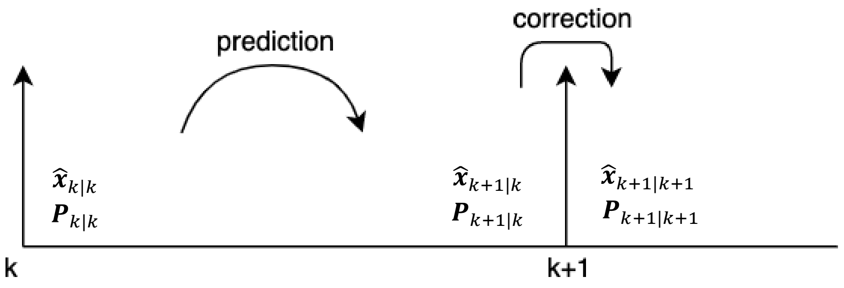
In order to estimate and predict the states of a non-deterministic, time invariant, discrete time system, in a state-space representation:

$$\begin{cases} \boldsymbol{x}_{k+1} = \boldsymbol{A}\boldsymbol{x}_k + \boldsymbol{B}\boldsymbol{u}_k + \boldsymbol{W}_k \\ \boldsymbol{y}_k = \boldsymbol{C}\boldsymbol{x}_k + \boldsymbol{V}_k \end{cases}$$

we will make use of an estimator:

$$\hat{\boldsymbol{x}}_{k+1} = \boldsymbol{A}\hat{\boldsymbol{x}}_k + \boldsymbol{B}\boldsymbol{u}_k + \boldsymbol{K}_k(\boldsymbol{y}_k - \boldsymbol{C}\hat{\boldsymbol{x}}_k)$$

where V_k and W_k are two vectors with stochastic entries that represent, respectively, the input sensor noise and the uncertainty of the model. We can suppose the entries of the vectors as normal distributed white noise variables with zero mean and covariance matrices, respectively, R and Q. The Kalman filter performs two operations: it predicts the next state using the modelled system and then it corrects the prediction with a filtering action.


prediction
correction
$\hat{\boldsymbol{x}}_{k|k}$
$\boldsymbol{P}_{k|k}$
$\hat{\boldsymbol{x}}_{k+1|k}$
$\boldsymbol{P}_{k+1|k}$
$\hat{\boldsymbol{x}}_{k+1|k+1}$
$\boldsymbol{P}_{k+1|k+1}$
k
k+1

The first step consists in predicting the next state knowing the modelled system dynamics

$$\hat{\boldsymbol{x}}_{k+1|k} = \boldsymbol{A}\hat{\boldsymbol{x}}_{k|k} + \boldsymbol{B}\boldsymbol{u}_k$$

and computing the uncertainty of the prediction expressed by the covariance matrix $\boldsymbol{P}$

$$\boldsymbol{P}_{k|k} = E[(\boldsymbol{x}_k - \hat{\boldsymbol{x}}_{k|k})(\boldsymbol{x}_k - \hat{\boldsymbol{x}}_{k|k})^T]$$
$$\boldsymbol{P}_{k+1|k} = E[(\boldsymbol{x}_{k+1} - \hat{\boldsymbol{x}}_{k+1|k})(\boldsymbol{x}_{k+1} - \hat{\boldsymbol{x}}_{k+1|k})^T].$$

Since we know that

$$\boldsymbol{x}_{k+1} - \hat{\boldsymbol{x}}_{k+1|k} = \boldsymbol{A}(\boldsymbol{x}_k - \hat{\boldsymbol{x}}_{k|k}) + \boldsymbol{W}_k$$

then, considering the mean value of the last expression

$$E[\boldsymbol{A}(\boldsymbol{x}_k - \hat{\boldsymbol{x}}_{k|k})(\boldsymbol{x}_k - \hat{\boldsymbol{x}}_{k|k})^T \boldsymbol{A}^T]$$

we can conclude that

$$\boldsymbol{P}_{k+1|k} = \boldsymbol{A}\boldsymbol{P}_{k|k}\boldsymbol{A}^T + \boldsymbol{Q}.$$

So, the whole prediction step is defined as

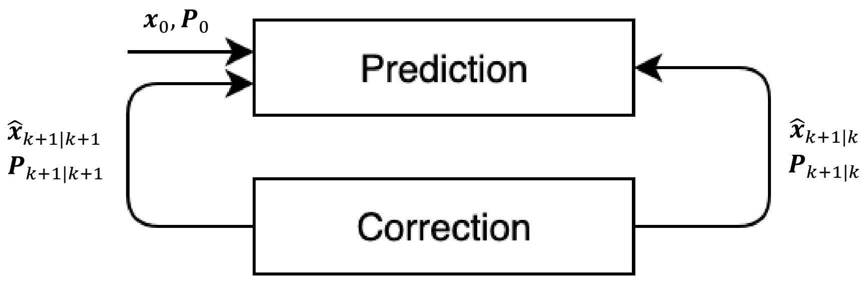
$$\boldsymbol{P}_{k+1|k} = \boldsymbol{A}\boldsymbol{P}_{k|k}\boldsymbol{A}^T + \boldsymbol{Q}$$
$$\hat{\boldsymbol{x}}_{k+1|k} = \boldsymbol{A}\hat{\boldsymbol{x}}_{k|k} + \boldsymbol{B}\boldsymbol{u}_k.$$

The second operation performed by the Kalman filter it is the correction step defined as

$$\hat{\boldsymbol{x}}_{k+1|k+1} = \hat{\boldsymbol{x}}_{k+1|k} + \boldsymbol{K}_{k+1}(\boldsymbol{y}_{k+1} - \boldsymbol{C}\hat{\boldsymbol{x}}_{k+1|k})$$
$$\boldsymbol{P}_{k+1|k+1} = (\boldsymbol{I} - \boldsymbol{K}_{k+1}\boldsymbol{C})\boldsymbol{P}_{k+1|k}$$

where K_{k+1} is the Kalman gain at step (k+1)-th, that can be computed as

$$\boldsymbol{K}_{k+1} = \boldsymbol{P}_{k+1|k}\boldsymbol{C}^T[\boldsymbol{C}\boldsymbol{P}_{k+1|k}\boldsymbol{C}^T + \boldsymbol{R}_k]^{-1}$$


$\boldsymbol{x}_0, \boldsymbol{P}_0$
Prediction
Correction
$\hat{\boldsymbol{x}}_{k+1|k+1}$
$\boldsymbol{P}_{k+1|k+1}$
$\hat{\boldsymbol{x}}_{k+1|k}$
$\boldsymbol{P}_{k+1|k}$

Models description

Mouse

To model the user-mouse interaction system, the bidimensional cinematic equation of a point without mass has been considered:

$$\begin{cases} x_{k+1} = x_k + v_{kx}\Delta T \\ y_{k+1} = x_k + v_{ky}\Delta T \end{cases}$$

where x_k, y_k are the Cartesian coordinates, v_{kx}, v_{ky} are the speed of the mouse pointer (at step k) and ΔT is the sampling time of the Kalman filter. Now, if we define:

$$\boldsymbol{x} = \begin{bmatrix} x \\ y \\ v_{kx} \\ v_{ky} \end{bmatrix}$$

as the state vector, we obtain a state-space system representation of the form:

$$\begin{cases} \boldsymbol{x}_{k+1} = \boldsymbol{A}\boldsymbol{x}_k + \boldsymbol{B}\boldsymbol{u}_k \\ \boldsymbol{y}_k = \boldsymbol{C}\boldsymbol{x}_k \end{cases}, \ \boldsymbol{A} = \begin{bmatrix} 1 & 0 & \Delta T & 0 \\ 0 & 1 & 0 & \Delta T \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \ \boldsymbol{B} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \ \boldsymbol{C} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

where here $\boldsymbol{x}_k$ represent the state vector and $\boldsymbol{y}_k$ represent the output vector at step k.

Ball

To model the mechanical system, the dynamic of a material point has been considered in this case and represented as a ball free to move on a plane according to the following equations:

$$\begin{cases} m\ddot{x} = F_x \\ m\ddot{y} = F_y \end{cases}$$

where F_x, F_y are randomly generated forces and where $\boldsymbol{x} = (x, y, \dot{x}, \dot{y})^T$ has been considered as state vector and $\boldsymbol{u} = \left(F_x, F_y\right)^T$ as input vector. Now we apply the zero order hold continuous-discrete conversion, obtaining:

$$\begin{cases} \boldsymbol{x}_{k+1} = \boldsymbol{A}\boldsymbol{x}_k + \boldsymbol{B}\boldsymbol{u}_k \\ \boldsymbol{y}_k = \boldsymbol{C}\boldsymbol{x}_k \end{cases}, \boldsymbol{A} = \begin{bmatrix} 1 & 0 & \Delta T & 0 \\ 0 & 1 & 0 & \Delta T \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \boldsymbol{B} = \begin{bmatrix} \frac{\Delta T^2}{2m} & 0 \\ 0 & \frac{\Delta T^2}{2m} \\ \frac{\Delta T}{m} & 0 \\ 0 & \frac{\Delta T}{m} \end{bmatrix}, \boldsymbol{C} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix},$$

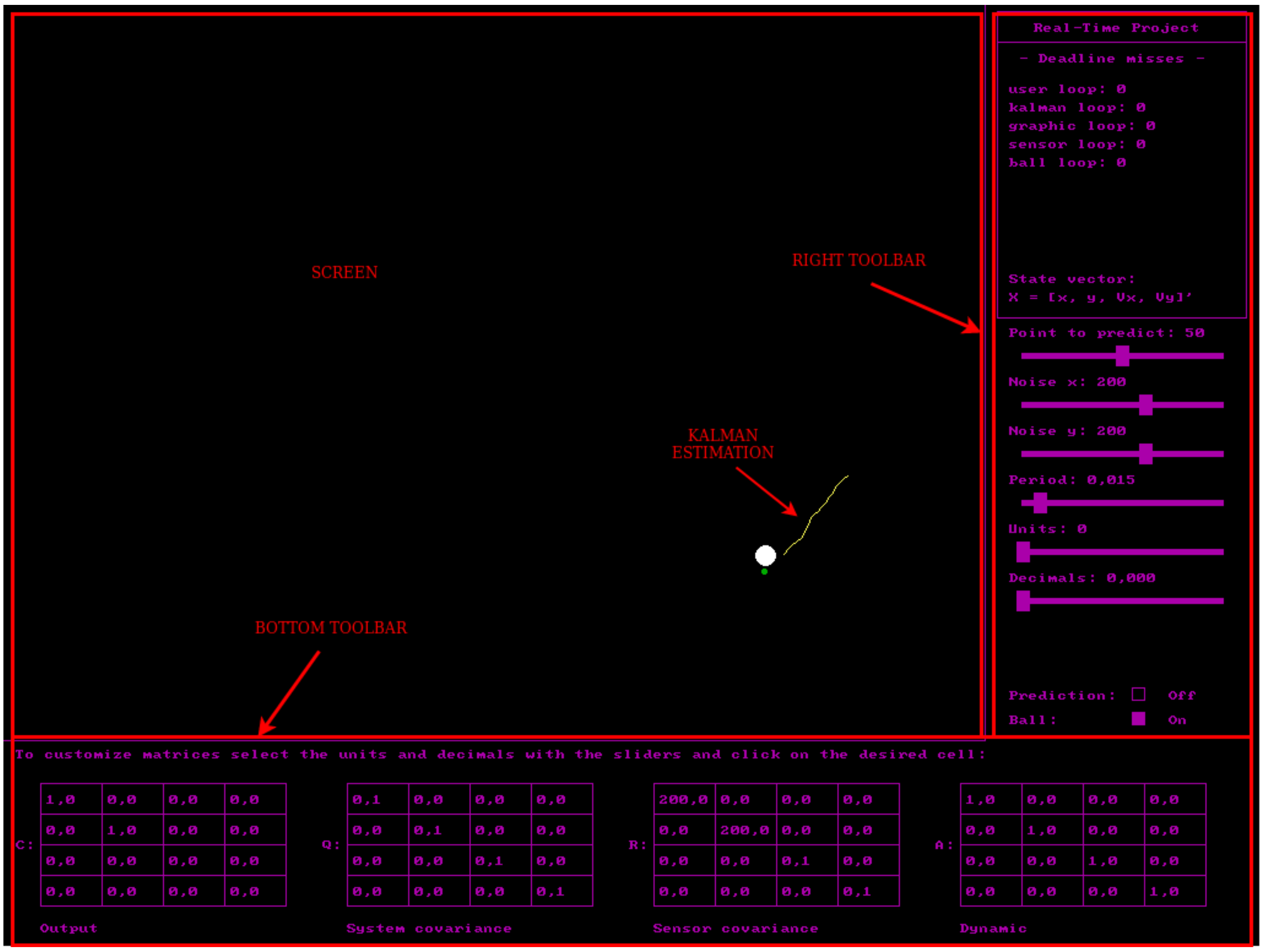
We point out that the forces F_x, F_y are created using two different stochastic variables, uniformly distributed in a range that goes from -10 to 10.

Graphic interface

The interface is divided into three graphical objects: the main frame, the right toolbar and the bottom toolbar. Inside the frame, the readings of the simulated sensor are indicated with green points, the output of the Kalman filter estimation with yellow lines, and the prediction, if requested by the user, is indicated with a red line.

On the right toolbar it is possible to see, inside a little console, the number of deadline missed by each task during the execution of the code and it is possible to tweak some parameters as, for example, enable or disable the Kalman filter prediction, change the number of predicted point, change the noise variance on the x or y coordinate and change the *ΔT* in the dynamic matrix A.

On the bottom toolbar the matrices C, Q, R and A are shown, all of them are customizable, by the user.



```
Real-Time Project
- Deadline misses -
user loop: 0
kalman loop: 0
graphic loop: 0
sensor loop: 0
ball loop: 0

State vector:
X = [x, y, Vx, Vy]'
Point to predict: 50
Noise x: 200
Noise y: 200
Period: 0,015
Units: 0
Decimals: 0,000
Prediction: ☐ Off
Ball:       ■ On
```

To customize matrices select the units and decimals with the sliders and click on the desired cell:

C: Output

1,0	0,0	0,0	0,0
0,0	1,0	0,0	0,0
0,0	0,0	0,0	0,0
0,0	0,0	0,0	0,0

Q: System covariance

0,1	0,0	0,0	0,0
0,0	0,1	0,0	0,0
0,0	0,0	0,1	0,0
0,0	0,0	0,0	0,1

R: Sensor covariance

200,0	0,0	0,0	0,0
0,0	200,0	0,0	0,0
0,0	0,0	0,1	0,0
0,0	0,0	0,0	0,1

A: Dynamic

1,0	0,0	0,0	0,0
0,0	1,0	0,0	0,0
0,0	0,0	1,0	0,0
0,0	0,0	0,0	1,0

Code overview

The entire project has been developed in C language with the support of the graphical library *allegro4.4* and the *pthread* library for real time programming. The project has been tested and executed on Ubuntu 19.04, Ubuntu 19.10 and Linux Mint 19.02. The project root directory has three sub-folders: *src* that stores *.c* and *.h* files, *bin* and *build* that are created at compilation time and contain binary files and object files respectively.

Libraries

To implement correctly the Kalman filter, it was necessary to create a library that allows to preform operations between matrices and vectors. *matrix* library contains the fundamental types of variable needed to create and manage matrices and vectors and to define all the basic operations between matrices:

- *struct matrice* is a new defined type of variable that contains the number of rows and columns and a *float* type pointer, pointing to the first element of the matrix.
- struct *systems* contains seven variables of type *matrice* (*A, B, C, Q, R, P, K*).
- struct *states* contains four variables of type *matrice* (x_k, x_pred, x_s, x_ball)

Moreover, the authors decided to create *ptask.h,* an additional library on the top of *pthread*, to simplify the management of periodic tasks. Here some functions that help threads management and periodic tasks creation are defined.

Tasks

When the code is executed *starter_task* launches four periodic tasks:

- *sensor_task* simulates the sensor behavior by reading the mouse/ball position and adding to it a normal distributed noise, then draws the position, obtained in this way, on the main frame (green dots).

- *kalman_task* performs the prediction and correction steps of the Kalman filter (as explained in the first section of this report) taking as input position the output of *sensor_task.*
- *user_task* manages and periodically checks user interactions for matrices customization and allows users to change the sliders position.

- *graphic_task* is the routine task that draws toolbars, the computed estimation of the Kaman filter (yellow line) and, if requested, the computed prediction (red line).

- *ball_task* simulates, if the relative button is enabled, the dynamic of a ball and draws its time evolution on the main frame. The initial position of the ball is randomly generated by *user_task* at the activation time.

Task	Period	Deadline	Priority
ball_task	10	10	30
sensor_task	15	15	25
kalman_task	15	15	20
user_task	10	10	15
graphic_task	33	33	10

Regarding the assigned priority values showed in the table above, we point out that those numbers are the results of empirical tests and system analysis. It was decided to execute *ball_task* every 10 ms to simulate as much as possible a continuous time system evolution. *sensor_task* and *kalman_task* are executed less frequently, with a period of 15 ms, to take into account the effect of discretization of the ball system. *user_task* has a period of 10 ms to allow responsive interactions between user and interface, and *graphic_task* has a period of 33 ms that is enough to smoothly simulate the evolution of the two different systems.

Shared data

The main shared data structures are:

Data	Type	ball	sensor	kalman	user	graphic
state	states	yes	yes	yes	yes	yes
sys	systems	yes	yes	yes	yes	yes
var_noise	float array	no	yes	no	yes	yes
fades	int array	no	no	no	yes	yes
p_to_predict	int	no	no	no	yes	yes
units	float	no	no	no	yes	yes
decimals	float	no	no	no	yes	yes

- *state* is a variable of type *states* that contains the state vectors of the systems, the last computed estimation and prediction of the Kalman filter and the sensor outputs.

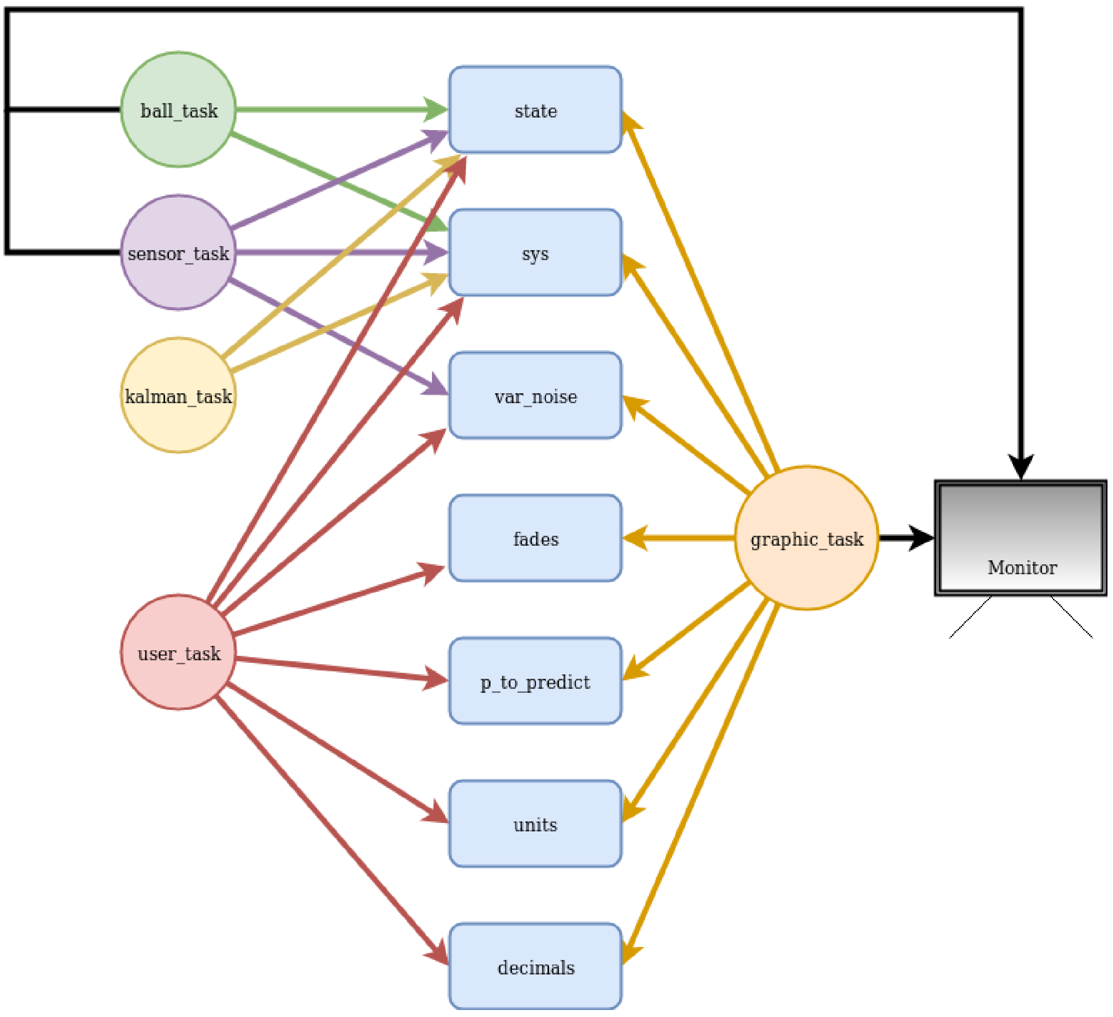
- *sys* is a variable of type *systems* that contains all the matrices for the state-space representation of all systems.

- *var_noise* is a variable of type *float* that defines the variance of the gaussian noise added to the mouse/ball position. The default value is 200 but it can be varied from the interface.

- *fades* is a set of arrays of variables of type *int* that defines the position of the sliders drawn by the graphic task.

- *p_to_predict* is a variable of type *int* that contains the number of points to predict and to draw on the main frame.

- *units* and *decimals* are *float* variables needed to customize matrices values.



Experimental results

While the tasks are running it is possible to tweak all the values of the matrices, the variance of the noise on both axis, the period T that influences the matrices A and B (it doesn't modify the period of *kalman_task* and *sensor_task*) and the number of point to predict. The best value for T, in order to better maintain the coherence with the sampling time, is the same value of *kalman_task* and *sensor_task* period. To have the best estimation, the better choice is to maintain the values R(1,1), R(2,2) as near as possible to the noise variance value; this means that the more we know about input noise, the better the filter performs.

The bouncing ball discontinuity is an unmodelled dynamic that could create problems with the estimation. This occurs when the uncertainty of the modelled dynamics (expressed by the matrix Q) is too low. In this case it is possible to observe that the estimation has to converge again to fix the error.

There are also some other parameters that can be varied in the *matrix.h* header file. The more interesting to tweak are:

- F_X, F_Y the upper limits in magnitude of the random forces.
- M the ball mass.
- T period of Kalman, sensor tasks and relative matrices.
- S period of ball task and relative matrices.
- CEIL_COUNT number of periods of the ball task in which the random force is constant.

During the test phase, the better results have been obtained with S = 10 ms, T = 15 ms, M = 1 kg, F_X = F_Y = 10 N due to better task performance and smooth ball visualization.

References

1. R.E. Kalman, "A New Approach to Linear Filtering and Prediction Problems", Transactions of ASM, Series D, Journal of Basic Engineering, Vol. 82, No, 1, 1960, pp 35-45.

2. The Single UNIX Specification, https://pubs.opengroup.org/onlinepubs/7908799/xsh/pthread.h.html.

3. Allegro Manual, https://www.allegro.cc/manual/4/api.