Plantilla para Proyectos en C/C++ del RAL

◁ *Estructura y Lógica del Programa* ▷

1 Objetivos

- Proveer una plantilla con la estructura y lógica básica para la implementación más rápida de algoritmos en proyectos del área de robótica, como algoritmos de control, estimación y otros de adquisición, procesamiento e interpretación de datos.
- Fijar un estándar mínimo para los proyectos del grupo RAL que garantice:
  - Orden y simplicidad $\Rightarrow$ claridad.
  - Modularidad y reusabilidad.
  - Portabilidad.

2 Descripción

La estructura general de toda solución de automatización requiere de tres pasos fundamentales:

1. Sensado: Medir ciertas variables que se desean controlar.
2. Procesamiento: Procesar los datos medidos para extraer información de interés.
3. Decisión/Acción: Verificar que las variables de interés cumplan un cierto objetivo fijado por el usuario, y de lo contrario, tomar acciones que ajusten las variables de interés del sistema.

Las tres etapas anteriores se repiten una y otra vez durante la operación del sistema. Cabe notar que en estas etapas están implicitos dos hechos. Primero, que el programa recoge o recibe datos, luego opera sobre estos y finalmente entrega ciertos resultados. Por otra parte, está el hecho de que existe un usuario que de alguna manera interactua con el sistema para fijar tanto objetivos como otros parámetros deseados de operación. En este sentido, la solución puede descomponerse en tareas o funciones de bajo nivel, aquellas encargadas de interactuar con el hardware, y otras de más alto nivel encargadas de la interacción entre el usuario y la máquina.
Por lo anterior podemos afirmar que existen por lo menos tres dimensiones principales con respecto a las cuales debe concebirse la solución:

- Entradas-Salidas: Consistente en las especificaciones de datos requeridos por el programa y los resultados que este debe generar.
- Temporal: Relacionada con la secuencia de ejecución de tareas necesarias para obtener los resultados.
- Niveles de abstracción de tareas: Relacionada con la separación de las funciones en grupos que abarcan desde la capa física hasta capas o niveles de abstracción mayor que implementan la interacción humano-máquina. En otras palabras, las capas inferiores implementan lo que en el ámbito del control automático se conocen como interfaces controlador-proceso, mientras que las capas superiores implementan las interfaces usuario-controlador en alguna forma como una GUI (*Graphical User Interface*), HMI (*Human-Machine Interface*), MMI (*Man-machine Interface*) o una simple CLI (*Command-Line Interface*).

Para una implementación eficiente, que garantice las características de claridad, modularidad, reusabilidad del código, es conveniente dedicar el tiempo que sea necesario a la definición clara y precisa de cómo sería el programa en cada una de las tres dimensiones anteriores. Esta inversión inicial de tiempo siempre es

recompensada, pues evita tener que reparar errores de diseño o estructuración que de lo contrario ocurren a medio camino. En las siguientes secciones se discuten brevemente las tres dimensiones tomando el programa plantilla PTZ como modelo de referencia. El documento concluye con una serie de recomendaciones para implementación de soluciones según el programa plantilla PTZ para proyectos RAL.

3 Especificaciones Funcionales

En ingeniería de sistemas y desarrollo de software, las *especificaciones funcionales* (llamadas en inglés *functional specifications, functional specs* o FSD = *functional specifications document*) corresponden al conjunto de documentos que describen el comportamiento requerido de un cierto sistema de ingeniería. Las espcificaciones funcionales permiten fijar un marco general de trabajo entre diversos programadores de un proyecto y describir lo que se requiere implementar. El documento típicamente se compone de las siguientes listas con descripciones de las propiedades de los datos y funciones que las componen:

1. **Inputs del Usuario**: Lista de datos entregados por el usuario a la aplicación. Algunos datos pueden ser variables modificables durante la ejecución a través de la interfaz de usuario, otros datos pueden ser parámetros establecidos al inicio de la ejecución de la aplicación.
2. **Inputs del Entorno/Proceso**: Lista de datos requeridos por la aplicación que son obtenidos desde el entorno.
3. **Outputs al Usuario**: Lista de datos o resultados que son entregados al usuario.
4. **Outputs al Entorno/Proceso**: Lista de datos que son entregados por la aplicación al entorno de modo que otras partes del entorno o sistema puedan realizar las acciones para las cuales fueron creadas.
5. **Funcionalidad**: Lista de funciones y comportamientos que debe poseer la aplicación.

Ejemplo Especificaciones Funcionales PTZ

Nombre de la Aplicación	ptz
Objetivos	
Abrir un archivo, secuencia de archivos, archivo de video o captura de una cámara y segmentar cada imagen o cuadro en base a características de color empleando las componentes de crominancia roja y crominancia azul del espacio de colores YCrCb. Además aplica un proceso simple de detección de bordes a la imagen segmentada.	
Inputs del Usuario	
Opciones de línea de comando	
`-h`	Prints some help.
`-nv`	Turns verbose off.
`-np`	Do not wait for user to press ESC when done.
`-nd`	Do not display images.
`-no`	Do not save the output.
`-e`	Filename extension [=*jpg*]
`-sq` *initial_value final_value*	Tells ptz to process a sequence of images starting at `filename<initial_value>.ext` and ending in `filename<initial_value>.ext` By default the sequence range is assumed to be between 0 and 10 with zero-padding = 3, i.e. 000, 001,..., 010.
`-zp` *num*	Pad filename suffix with num zeros.
`-fps` *num*	Frames per seconds [=*max_fps*].
GUI, Mouse y Teclado	
ESC	quit the program.
Mouse-L	show position and intensity values of selected pixel.
Mouse-R	select region.
Mouse-R+CTRL	draw on buffer.
p	Process current image using loaded or precomputed data.
a	Add class.
r	Remove current class.
l	List existing classes.
u	Calculate feature vector for current region and update the class parameters.
b	Blank current class data.
v	Fill current class data with random values.
<	Decrease current class.
>	Increase current class.
h	Toggle process holding (temporarily disables/enables processing).
g	Get class data from cpd.txt.
s	Save class data to cpd.txt.
t	Test basic functions.
Threshold slidebars	Segmentation threshold and edge thickness threshold.
Inputs del Entorno	
Image or video	Image file or video sequence from AVI file or USB camera.

Outputs al Usuario	
Segmentation	Image containing the segmentation of the input into the differnt classes.
Edges	Image containing the edges of the segmented input.
Outputs al Entorno	
Segmentation map	Image containing the segmentation map (result).

Funcionalidad

1. La aplicación detecta en forma automática el tipo de input (archivo único de imagen, secuencia de archivos de video, archivo de video AVI o captura directa de cámara USB).
2. La imagen puede ser segmentada según las características de color seleccionadas por el usuario. Para esto el usuario primero debe seleccionar una región de interés con Mouse-R+CTRL, luego añadir una clase con 'a', y actualizar los valores de la clase actual con 'u'. El procesamiento se inicia cuando el usuario presiona 'h'.

Referencias sobre Especificaciones Funcionales

[1] Wikipedia. Functional Specifications. Versión 28 de octubre, 2008.
`http://en.wikipedia.org/wiki/Functional_specifications`

[2] Allen Smith. Functional Spec Tutorial: What and Why. Visitada 16 de diciembre, 2008.
`http://www.mojofat.com/tutorial/`

4 Estructura Temporal

La estructura temporal se refiere a la secuencia lógica de ejecución. Esta puede describirse mediante diagramas de flujo o pseudocódigo. Los diagramas de flujo pueden resultar más explicativos que los pseudocódigos cuando se requiere describir el flujo de datos, pasos de un proceso o sistema con muchas etapas de decisión. En cambio los pseudocódigos resultan más eficaces para describir funciones o algoritmos, los cuales por lo general contienen más pasos de cálculo que etapas de decisión.

Ejemplos de Pseudocódigo

Para ilustrar el uso de pseudocódigos y diagramas de flujo se presentan a continuación algunos procesos computacionales. El algoritmo 1 corresponde a una descripción en *pseudocódigo de alto nivel* de los pasos principales llevados a cabo en el programa PTZ. Esta estructura corresponde a la estructura general de toda aplicación de automatización, como se explicó en la sección 2. La secuencia de tareas principales realizadas por el programa PTZ también puede describirse empleando un *pseudocódigo de bajo nivel* como se muestra en el algoritmo 2. La diferencia entre un pseudocódigo de alto nivel y uno de bajo nivel yace en que el primero está escrito empleando oraciones similares a las del lenguaje habitual de las personas junto con fórmulas matemáticas, mientras que el pseudocódigo de bajo nivel emplea términos y un lenguaje que asemeja un lenguaje de programación. En un pseudocódigo de bajo nivel es habitual observar términos que representan las bifurcaciones condicionales (`if-then-else`) y otros términos que usualmente se emplean para *loops* o bucles de programa (`for-from-to-do`, `while-do`, `do-while` o `repeat-until`). Otro ejemplo de pseudocódigo de bajo nivel se muestra en el algoritmo 3. Este último algoritmo corresponde a la implementación de la función `ComputeSegmentation` del programa PTZ. Al comparar el pseudocódigo con el código original en C/C++ de la función, podrá notar que el pseudocódigo es bastante más fácil y rápido de leer que el código fuente.

Finalmente, el pseudocódigo de bajo nivel de un proceso general de automatización se muestra en el programa 4. Cabe destacar que en este caso el pseudocódigo de bajo nivel hace uso de una sintaxis similar a la del lenguaje C/C++. Por ejemplo, el inicio y el fin de cada función están definidos por los delimitadores {, }, respectivamente. Por otro lado, los comentarios emplean // y las funciones indican el tipo de dato que retornan.

¿Cuándo utilizar pseudocódigo de alto o de bajo nivel?

Utilice pseudocódigo de alto nivel para describir procesos muy generales o procesos en los que las etapas de decisión son pocas o son elementos secundarios del proceso. Un pseudocódigo de alto nivel es ideal para entregar una visión general de la secuencia de pasos que se debe realizar, sin entrar en los detalles de cada paso. Los pseudocódigos de bajo nivel permiten explicar de mejor manera algoritmos y funciones. Los pseudocódigos de bajo nivel no ofrecen mayor ventaja para explicar una secuencia general de pasos de un programa o aplicación. Al emplear pseudocódigo de bajo nivel, procure utilizar un lenguaje genérico como en los algoritmos 2 y 3. Existen herramientas en LaTeX, como el paquete `algorith2e` que permiten escribir pseudocódigo de bajo nivel estándar (vea el código fuente de este documento). Evite utilizar pseudocódigo de bajo nivel que recurra a términos y sintaxis de un lenguaje de programación específico. Recuerde que el propósito del pseudocódigo es que sea entendido por cualquiera y pueda ser implementado en cualquier lenguaje de programación. Utilice una sintaxis específica a un cierto lenguaje solo cuando sea necesario referirse a programas escritos en ese lenguaje, como en ejemplos de programación o aplicación del lenguaje particular.

Algorithm 1: Main Program Structure (High-level Pseudocode)

input : Image sequence
output: Segmented image using YCrCb color features

1. Initialize data structures
2. Parse command line parameters and update data structures
3. Get the input image
   Depending on the context some of the typical actions may include commands with names such as: `Allocate/Create/Open/Load/Read/Get/Retrieve/Grab/Capture`
4. Process image
5. Store the results
   Depending on the context some of the typical actions may include commands with names such as: `Save/Write/Set/Put/Close/Release/Free/Clear/Destroy`
6. Show/display results
7. Handle keyboard inputs from user
8. Repeat 3-7 until some ending condition is met, skipping 6 and 7 if no user interaction through CLI or GUI is required in order to save time.

Algorithm 2: Main Program Structure (Low-level Pseudocode)

input : Image sequence
output: Segmented image using YCrCb color features

—— *Initialization* ——;
`InitCmdLineStruct(&CmdLineParam);`
`ParseCmdLine(&CmdLineParam);`
`CheckInputType(&CmdLineParam);`
`CreateInputContainer;`
if &CmdLineParam.display = TRUE **then**
  `CreateGUI;`
end
`InitProcessParam(&ProcessParam);`

—— *Begin Process* ——;
while (capture)∧(run) **do**
  &img ← `GetImage();`
  &img ← `ProcessImage(&img, &ProcessParam);`
  `StoreImage(&img);`
  if &CmdLineParam.display = TRUE **then**
    `DisplayImage();`
  end
  `ProcessKeyboard();`
end

Algorithm 3: ProcessImage (Low-level Pseudocode)

input : &img (input image), &ProcessParam (process parameters)
output: &img (segmented image)

&imgaux ← &img;

—— *ComputeSegmentation* ——;
for $row = 0$ **to** &img.height **do**
  for $col = 0$ **to** &img.width **do**
    for class = 1 **to** &ProcessParam.Nclasses **do**
      if ‖`ConvertToYCrCb`(&imgaux(row,col)) − &ProcessParam.FeatureVector(class)‖ <Threshold **then**
        &img ← class;
      end
    end
  end
end

Control Program

Procedure *double* sensor(*void*)

```
{
    // Initialize/Connect/Synchronize Sensor
    Open(SensorParam);
    Connect(SensorParam);
    Enable(SensorParam);

    // Retrieve sensor data
    measurement = Read(SensorParam); // Other names for this function are Get/Load

    // Close sensor
    Disable(SensorParam);
    Disconnect(SensorParam);
    Close(SensorParam);
    return measurement;
}
```

Procedure *void* actuator(*double* value)

```
{
    // Initialize/Connect/Synchronize Actuator
    Open(ActuatorParam);
    Connect(ActuatorParam);
    Enable(ActuatorParam);

    // Set actuator
    Write(ActuatorParam,value); // Other names for this function are Set/Put

    // Close actuator
    Disable(ActuatorParam);
    Disconnect(ActuatorParam);
    Close(ActuatorParam);
}
```

Procedure *void* main(*int* argc, *char*** argv)

```
{
    // ------ Define constants and parameters ------
    double measurement;
    double value;
    struct {...} SensorParam;
    struct {...} ActuatorParam;
    ...

    // ------ Process Loop ------
    while (getch() != 0x1B) // Wait till ESC is pressed
    {
        // ------ Read sensor ------
        measurement = sensor(SensorParam);
        printf("Measured value [unit]: %f", measurement);

        // ------ Write actuator ------
        value = Process(measurement);
        actuator(ActuatorParam,value);

        // ------ Read keyboard ------
        ProcessKeyboard();
    }
}
```

Ejemplos de Diagramas de Flujo

El diagrama de flujo del programa plantilla PTZ se muestra en las figuras 1 y 2. Este diagrama de flujo corresponde a los pseudocódigos 1 o 2. La comparación del diagrama de flujo con los pseudocódigos revela ciertas diferencias que no están indicadas en forma explícita en el pseudocódigo. Por una lado, podemos ver que el diagrama de flujo toma parámetros iniciales de la línea de comando, inicializa las estructuras de datos apropiadas, luego determina si el input a ser procesado es una imagen de archivo o un video de entrada. A continuación verifica si se deben o no desplegar las pantallas con las imágenes y los resultados procesados, es decir si debe o no desplegar la interfaz gráfica o *graphical user interface* (GUI). El proceso, que continua en la figura 2, inicializa otras variables propias de la etapa de procesamiento de las imágenes y dependiendo de si el usuario indicó o no la opción de generar un resultado de manera "verbosa", la aplicación informará o no al usuario sobre lo que esta se encuentra ejecutando. En el caso en que el procesamiento del input es verboso, las imágenes se despliegan y el usuario tiene la opción de interactuar con la aplicación a través del teclado. Específicamente, el usuario puede activar o desactivar la realización del proceso. Cuando el *holding* esta desactivado (en *off*), la aplicación procesa las imágenes. En otras palabras, cuando *holding off* es verdadero la aplicación deja un modo tipo *stand-by* en el cual las imágenes que se obtienen se despliegan al usuario sin ser procesadas. Cuando *holding* no está en *off*, se tiene *holding on*, caso que corresponde a la rama izquierda de la bifurcación del programa. En ambos casos, *holding off* u *on*, el paso siguiente es verificar si el usuario presionó la tecla `ESC` para terminar la aplicación o si se acabó la secuencia de imágenes, de lo contrario la aplicación sigue procesando imágenes indefinidamente en el *loop* que se cierra antes de la verificación de la condición de *holding*.
La figura 3 muestra otro ejemplo de diagrama de flujo. Este diagrama corresponde al algoritmo de procesamiento de segmentación por color, cuyo pseudocódigo se presentó en el algoritmo 3. Una comparación cuidadosa del pseudocódigo del algoritmo 3 y el diagrama de flujo 3 permite verificar que efectivamente son equivalentes y hay dos bucles `for-from-to-do` anidados. Sin embargo, es posible concluir que en este caso la descripción del proceso de segmentación mediante un diagrama de flujo es menos eficiente, pues es más extensa y no se comprende tan rápidamente como el pseudocódigo que requiere de unas pocas líneas.

¿Cuándo utilizar un diagrama de flujo y cuándo un pseudocódigo?

Como se mencionó, usualmente conviene utilizar diagramas de flujo para describir la estructura general de una aplicación, mientras que algoritmos y funciones específicas se pueden expresar de manera más concisa mediante pseudocódigo. Para crear diagramas de flujo en LaTeX existen diversas herramientas. Se recomienda consultar para mayores detalles y ejemplos el tutorial de la referncia [1] de esta sección.

Referencias sobre Diagramas de Flujo y Pseudocódigo

[1] Miguel Torres Torriti. Creating flowcharts in LaTeX Using the *pic* Language. 21 de diciembre, 2008. Disponible en:
`http://www.ing.puc.cl/ mtorrest/downloads.htm`

[2] Wikipedia. Flowchart. Visitada 17 de diciembre, 2008.
`http://en.wikipedia.org/wiki/Flowchart`

[3] Wikipedia. Algorithms on Wikipedia. Visitada 17 de diciembre, 2008.
`http://en.wikipedia.org/wiki/Wikipedia:Algorithms_on_Wikipedia`

[4] Wikipedia. Pseudocode. Visitada 17 de diciembre, 2008.
`http://en.wikipedia.org/wiki/Pseudo-code`

[5] Wikipedia. Pidgin Code. Visitada 17 de diciembre, 2008.
`http://en.wikipedia.org/wiki/Pidgin_Algol`

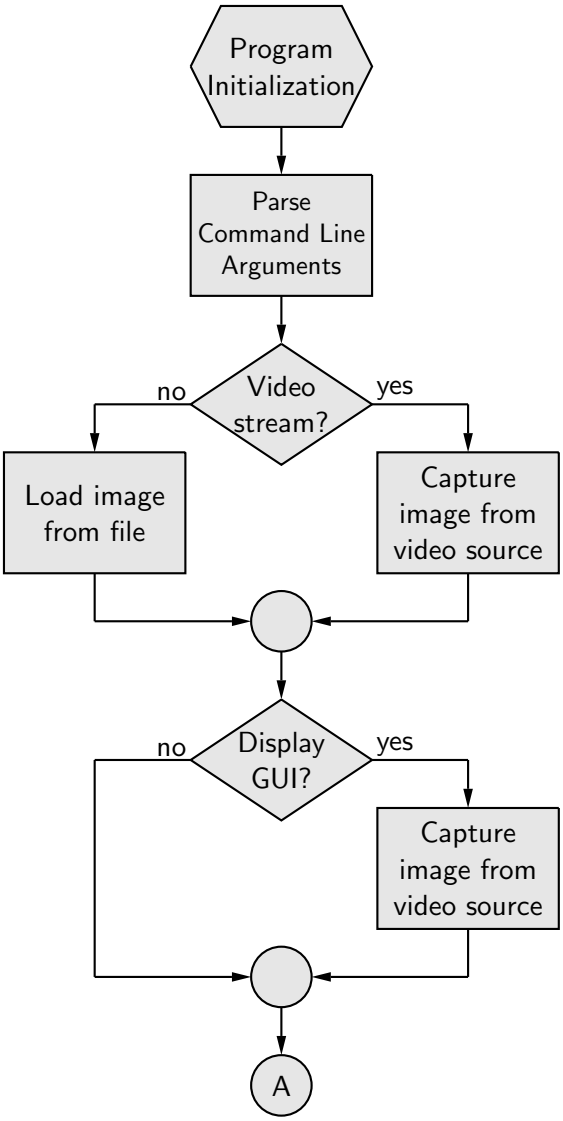

Figura 1: Diagrama de flujo del programa PTZ (continua en la figura 2).

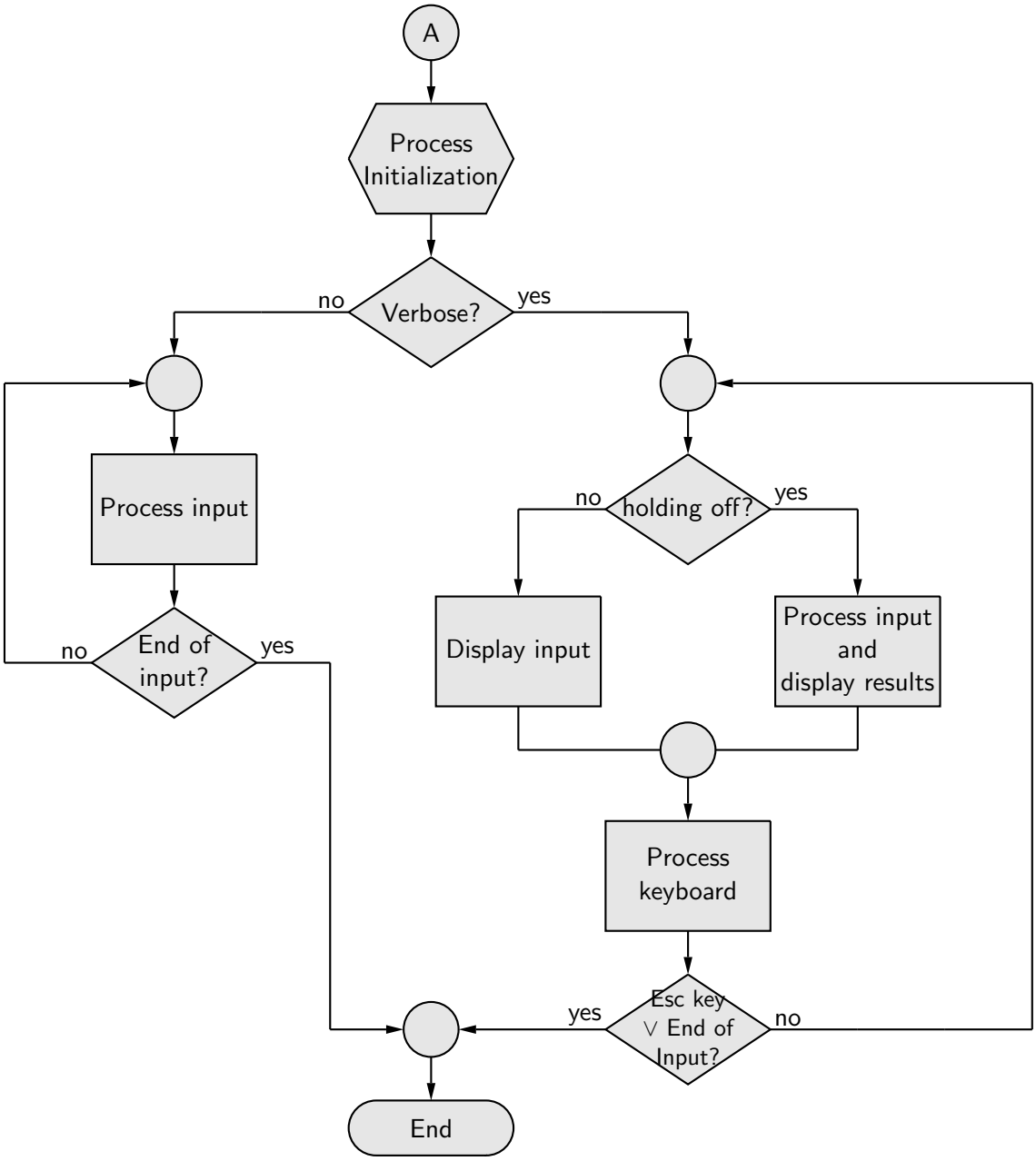

Figura 2: Diagrama de flujo del programa PTZ (continuación de la figura 1).

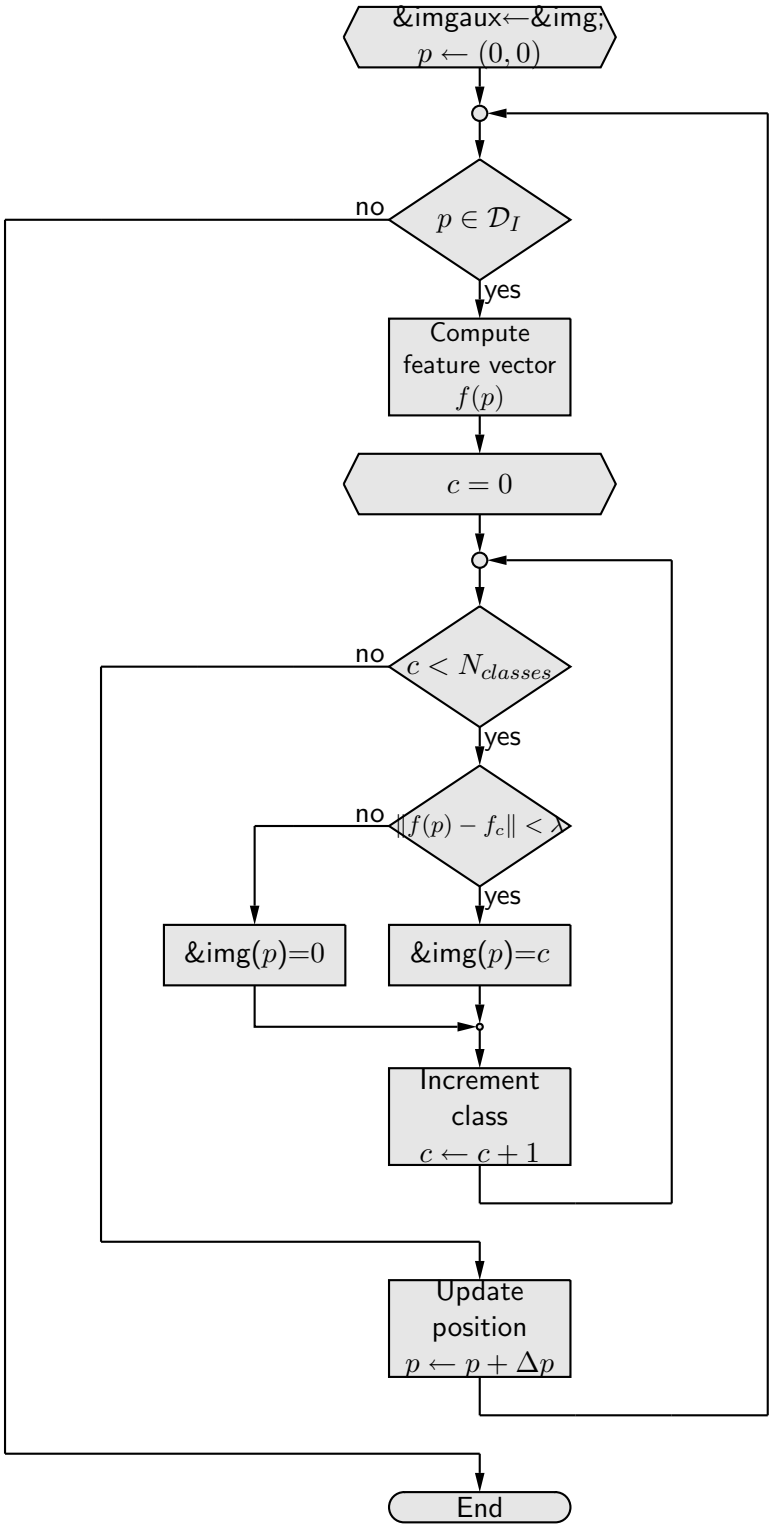

Figura 3: Proceso de segmentación de imágenes por color.

5 Niveles de Abstracción

La mayor parte de las aplicaciones se puede descomponer al menos en las siguientes categorías de funciones:

- Funciones Utilitarias Básicas / Basic Utility Functions: IntiCmdLineStruct, ParseCmdLine, etc.
- Funciones Matemáticas Básicas / Basic Math Functions: mvectoralloc, mmatrixalloc, etc.
- Funciones del Proceso / Process Functions: AddClass, ComputeSegmentation, etc.
- Funciones GUI / GUI Functions: on_mouse, proceso de teclado, etc.

La lista anterior incluye ejemplos de funciones del programa PTZ que pertenecen a cada categoría. Uno podría pensar en otras descomposiciones, por ejemplo, funciones de soporte de hardware, funciones de control de acceso a discos, funciones gráficas, funciones de acceso a bases de datos, funciones de comunicación entre equipos, etc. Lo importante es establecer una estructura que permita agrupar funciones en archivos que puedan reutilizarse. Así la clasificación anterior permitiría tomar las funciones básicas de servicio y emplearlas en otro proyecto o emplear las funciones matemáticas en otro programa, tal como si fuesen funciones de una librería o *toolbox* de software.

Ejemplo de Diagrama de Niveles de Abstracción

La figura 5 muestra los distintos niveles de abstracción del programa PTZ. Las funciones que aparecen con asterisco (*) no necesariamente están implementadas como un procedimiento, sino como líneas dentro del código, o son funciones con un nombre distinto al que se presenta en la figura 5 con el fin de facilitar la comprensión. Es posible ver en el esquema de la figura 5 que existen funciones en distintas categorías que entregan información al usuario a través de la pantalla. Si bien uno podría pensar que estas funciones pertenecen a la categoría GUI, considerando el hecho de que el despliegue de datos en pantalla realizado por estas funciones es básico (directo al terminal o ventana de comando) y más bien son funciones auxiliares de su categoría (permiten visualizar los contenidos de estructuras de datos como vectores o matrices), su clasificación en una categoría distinta a GUI es más apropiada. En la categoría GUI solo debieran aparecer funciones que soportan la creación y gestión de la interfaz gráfica de usuario, las cuales habitualmente requieren de librerías propias al entorno de programación utilizado.

6 Recomendaciones Finales

1. Utilice la plantilla PTZ para sus proyectos y aproveche su estructura. En principio solo requiere reemplazar las *Funciones del Proceso* del programa PTZ (ver figura 5) por sus propias funciones. Si adicionalmente emplea su propia librería de funciones matemáticas, como la GNU Scientific Library (GSL), entonces deberá eliminar las funciones del grupo de *Funciones Matemáticas Básicas* del programa PTZ.

2. Antes de empezar a escribir código en el computador dedique algo de tiempo a pensar como implementará la solución en términos de las tres dimensiones señaladas en este documento. Genere sus especificaciones funcionales, su diagrama de flujo, pseudocódigos y esquemas con los niveles o grupos de abstracción. Este tiempo invertido lo recuperará rápidamente y verá que con la experiencia este enfoque resultará ser el camino natural y más expedito para hacer las cosas bien a la primera.

3. No olvide comentar su código. Ciertas soluciones a las que uno llega después de pensar mucho resultan obvias o triviales, y con frecuencia su razón o la manera de llegar a ellas se olvida, dificultándose así la posterior comprensión y reutilización del código.

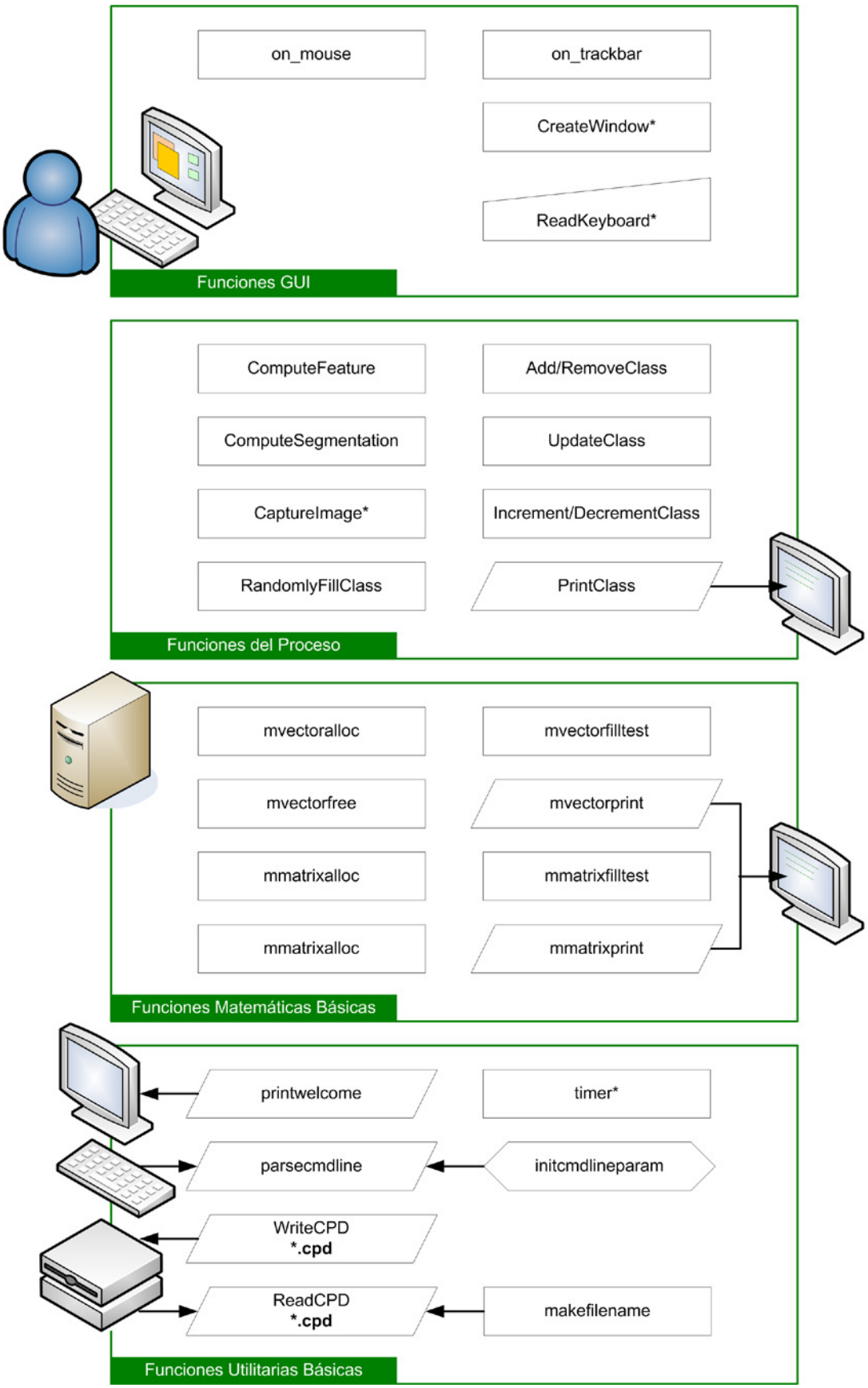

Figura 4: Niveles de abstracción.