Bruno Lemos 98221, Tiago Marques 98459

1 CAB - Cyclic Asynchronous Buffers

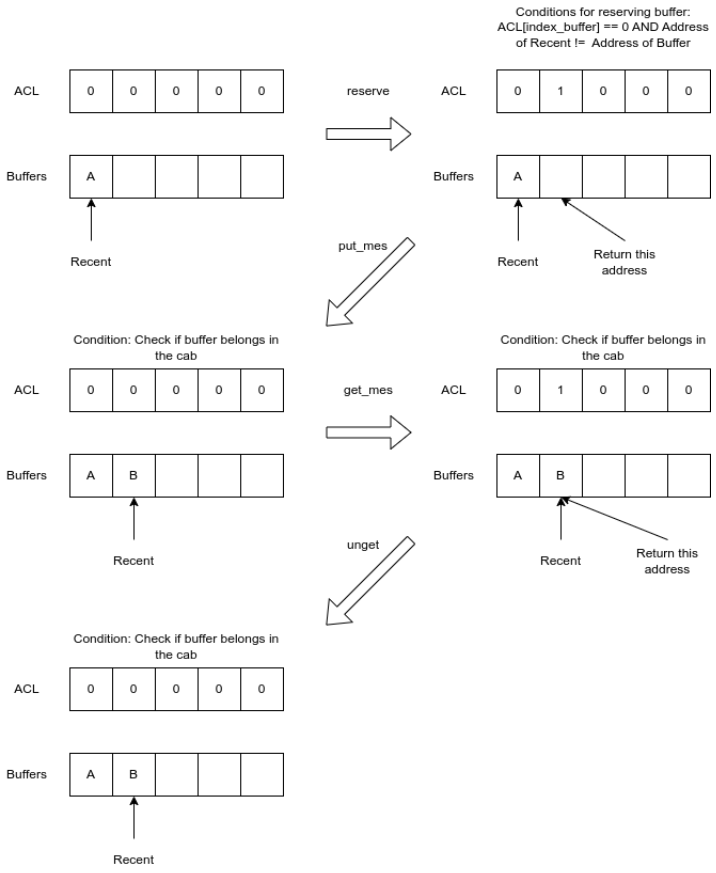

Figure 1: Diagram of operations

reserve

1 - Look for available buffer, ACL[buffer_index]==0 AND Buffer != Recent.
2 - Add 1 to the status of buffer, ACL[buffer_index] += 1 .
3 - Returns address of the buffer.

put_mes

1 - Check if the cab_id is valid
2 - Check if the buffer belongs to the cab
3 - Update the recent pointer to the buffer with the most recent message
4 - Subtract 1 to the status of the most recent message

get_mes

1 - Check if the cab_id is valid
2 - Add 1 to the status of the most recent message
3 - Returns address of the buffer with the most recent image

unget

1 - Check if the cab_id is valid
2 - Check if the buffer belongs to the cab
3 - Subtract 1 to the status of the most recent message

Super Importante note : In the first deliver of the project, we had a function with the name release, that was supossed to have the name unget. In this delivery of the project we corrected this mistake.
Note: During the processes described above, we use a semaphore to guarantee exclusivity in accessing and/or modifying variables within the CAB. Also the 'Buffers' in the image is an array of buffers and the 'ACL' is an array of integers with the purpose of keeping the status of the buffers.

2 Image Processing Tasks

2.1 Flow of Task Execution

All image processing tasks, Near obstacle, Obstacle Counting and Orientation and Position, are sporadic tasks that are activated when there is a new image available in CAB. To do this we use semaphores to control the access.

It is also important to note that for each new image received, it is only processed once per thread. In this case, each image is processed 3 times, 1 time for each processing thread. After each activation of these image processing threads, another thread (Output Update) is activated.

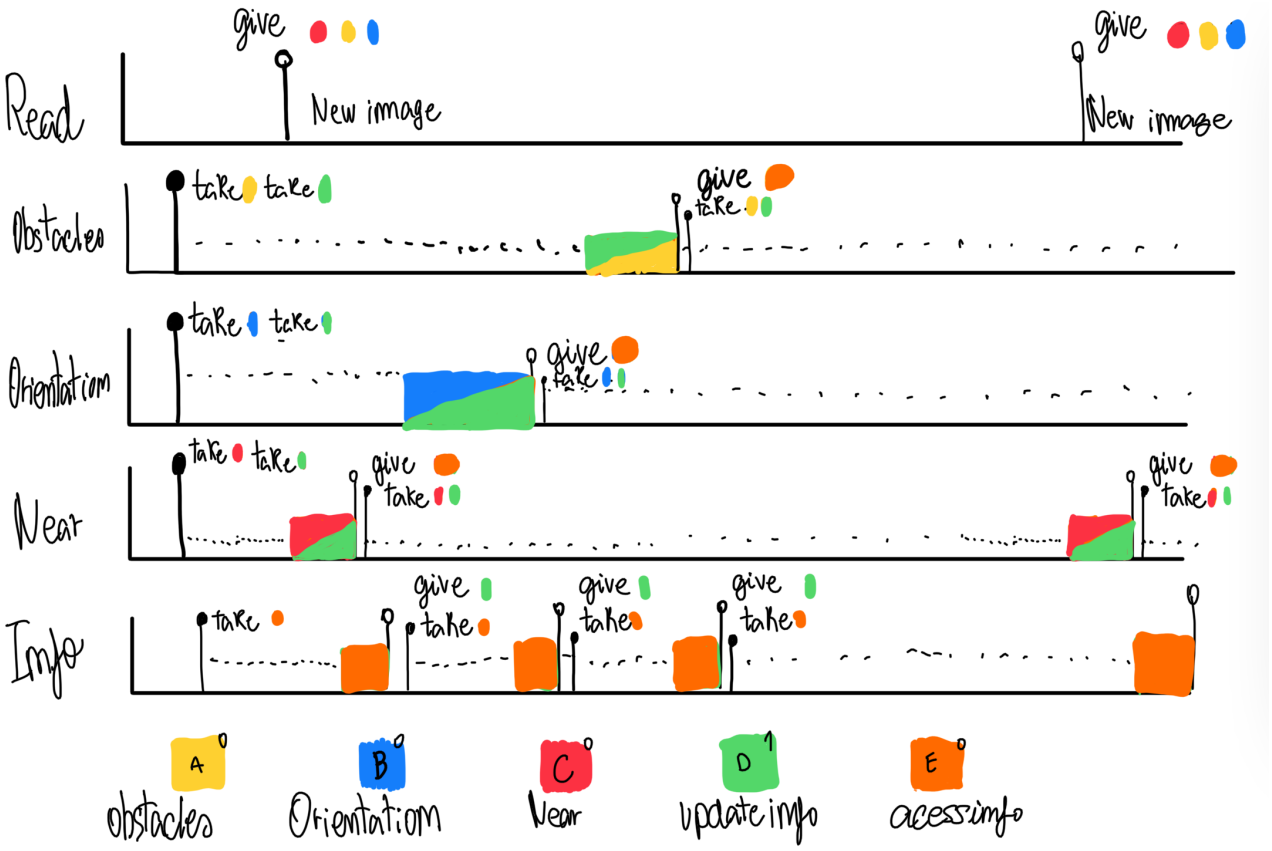

Figure 2: Thread processing

2.2 How we interpret the image

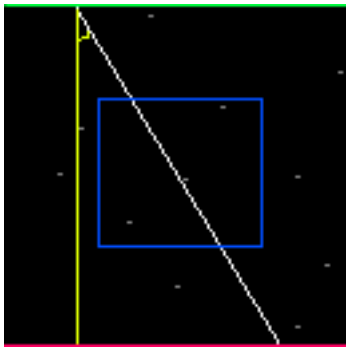

Figure 3: Example of a image

Position
In our case the position calculated in thread Orientation and postion and it's based on how close we are to the end of the green line (shown in the figure above). The closer the guide line pixel is to the left, the closer the position percentage value is to 0, otherwise the closer the line pixel is to the right, the greater the position percentage value.
Angle
Regarding the angle, it is obtained according to the figure above. If the point described on the green line is further to the left than the red line then the angle is negative, otherwise the angle is positive. There is also the special case that they have the same x coordinate, in which case the angle is 0.
Near Obstacles
The search area for near obstacles is defined by the blue area described in the figure. The value is a boolean that represent if there is any pixel with the value different from 0x00 (color of the floor) or 0xFF (color of the guideline) inside that area.
Count Obstacles The whole image is searched for obstacles. For an obstacle to be found and added to the number of obstacles in the image we need to have 2 consecutive pixels with the value 0x80 in the horizontal, therefore there are no vertical obstacles.

3 Temporal characterization of the Tasks

3.1 Caracterization

The near execution took 1/2 ms, while orientation took 1/2 ms, counting took 4/5 ms, and thread information took 9/10 m

3.2 WCET

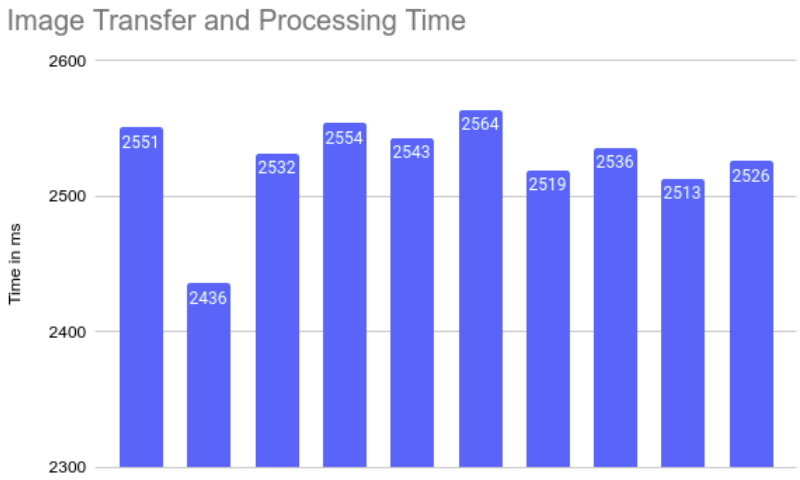

Figure 4: Times for transfering and processing an Image

We measure the WCET by measuring the times it takes the program to transfer an image and process it. We know that this method has its disadvantages such as the accuracy of the given WECT can be affected by the state of data within the microcontroller, variations in the hardware environment, as well as other factors such as interference from other processes running on the same system. Despite this, we chose to use it because it is easy to implement and also provides accurate and detailed information about the execution time of a program.

The image above shows a small sample of the values taken for the execution time of the image transfer and processing. Looking at the figure above, we can see that 2564 ms was the worst execution time, so we can say with some certainty that WCET must have an approximate time of that value.

4 Description, Results and its analysis

4.1 Cab Utility Tests

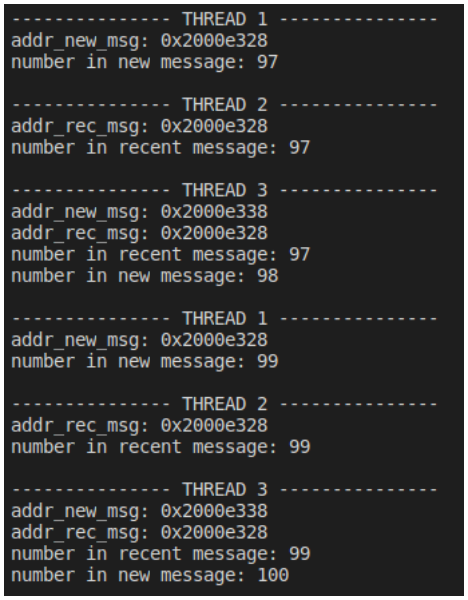

Figure 5: Execution of cab test

Execution 1 of thread 1: We reserve a free buffer in the cab (address 0x2000e328) and make the number 97 the most recent.
Execution 1 of thread 2: We request to most recent message from the cab (address 0x2000e328) and read the number 97. Then we unget the message.
Execution 1 of thread 3: We reserve a free buffer in the cab (address 0x2000e338) and also request the most recent image (address 0x2000e328). Then we proceed to make the number 98 the most recent message.
Execution 2 of thread 1: We reserve a free buffer in the cab (address 0x2000e328) and make the number 99 the most recent. **Why the address 0x0x2000e328 again?** When looking for a free buffer, we iterate through the ACL until we find a buffer with a status equal to 0. This iteration process starts with ACL[0] until ACL[num_of_messages], therefore, as buffers[0] has the address 0x2000e328 and its corresponding state is found in ACL[0], when the buffer is free it is allocated again.

4.2 Thread Execution Order

The thread that detects if there are near obstacles is the first to be executed, then the thread that calculates de position and the angle and finnaly the thread that computes the number of obstacles in the image. Note between the execution of these threads the thread that outputs the

Thread Info is executed whenever a processing thread is executed.

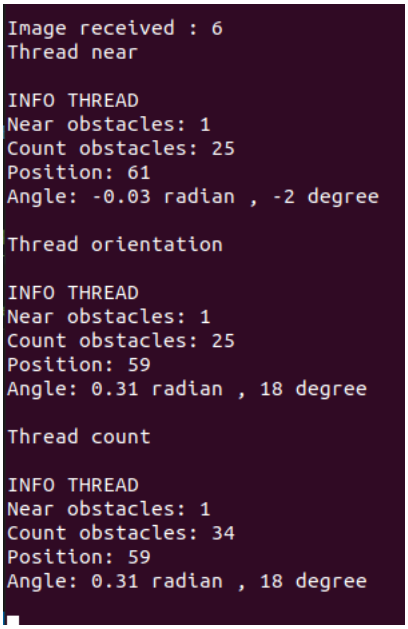

Figure 6: Thread executong order

5 Additional work

5.1 Generate images

We create a python script ("generate_image.py") to generate random images according to the specifications. This script generates png and raw images. The png iamges are for debugging in a more visual way and the raw are used by the computer application. This aplication have parameters to create versatility: size of image, max number of images, max number of obstacles, color of obstacles, color of background and color of guideline.

5.2 Comunication Protocol

To send images in such a way that there is no loss of information from the computer to the board, we use a handshake protocol. Packets of 64 bytes are sent and the board sends an ACK with byte '0x24'.

In the computer program part, all received bytes are filtered:
- '0x24' : ACK, send another 64bytes
- others : Strings, so print it