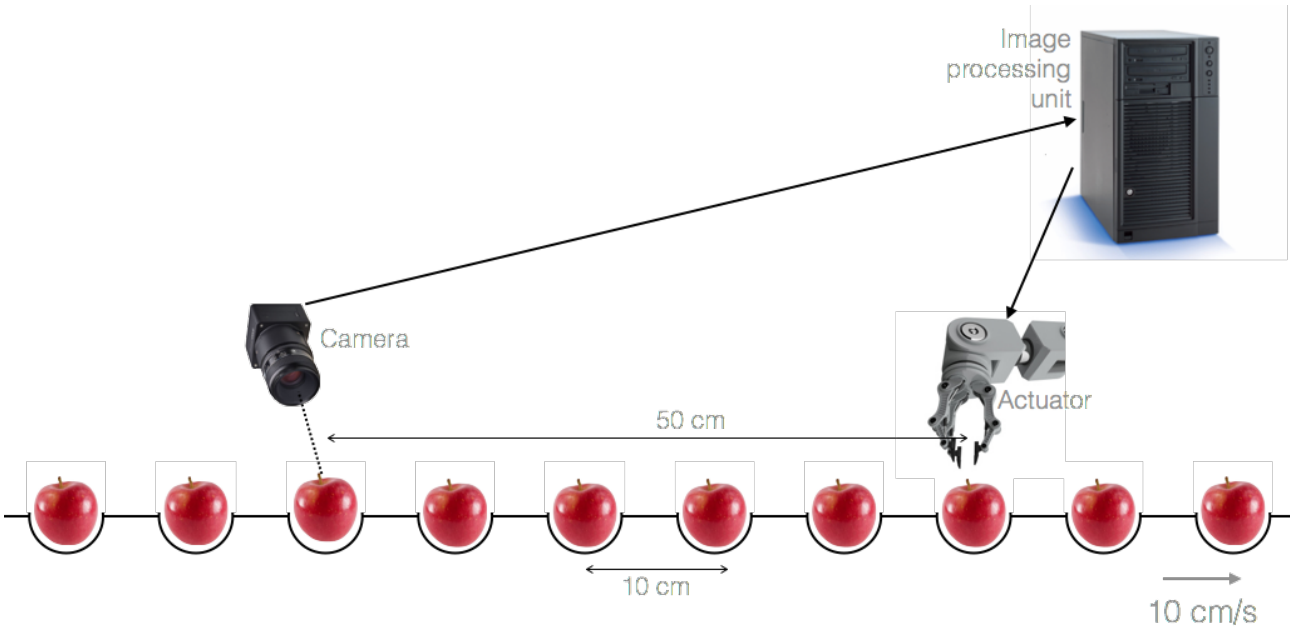
EE5903 – CA1 Exercise

Individual CA – AY 2015/2016 Semester II

Problem Statement

You have been engaged as a software architect and developer by a large organic apple farm to help develop a system to improve the packing process for the apples. Since the organic farm avoids insecticides and pesticides, a sizable percentage of the apples produced tend to be damaged. A recent sample yielded about 10% bad apples. Apples are packed in baskets of 24 for shipping. If one or more of the apples in a basket is bad, all apples in that basket spoil during transit, and the basket has to be discarded. Hence it is financially important to identify and discard bad apples before packing.

The packing of apples is an automated process, with 10 concurrent packing lines operating continuously (24 hours per day). The conveyer belt that carries the apples moves at 10 cm/s and has slots to hold apples every 10 cm. A camera is installed to take photos of the apples as they pass on the belt. For an acceptable photograph, the camera has to be activated within ±1 cm of the apple passing under it. The photo taken by the camera is to be fed to an image processing unit that identifies if the apple is good or bad. The image processing unit can only process one image at a time, and has a processing time that is exponentially distributed with a mean of 990 ms. Once started, the image processing cannot be interrupted. If the apple is bad, an actuator can be activated to discard that apple as it passes the location of the actuator 50 cm from the location of the camera. The actuator must be activated within ±2 cm of the bad apple passing its nominal location, for that apple to be discarded. To estimate the financial impact of the system you are developing, you may assume that each good apple is sold for about $1.00, and ignore the cost of transportation and distribution.



The manufacturers of the camera, actuator and the image processing unit have made their application programming interface (API) available to you on UNIX as a library that can be used by a C program. The API also includes a test harness that allows you to measure the performance of your software by simulating the apple packing line with the statistics collected during the recent sampling exercise. The documentation of the API is attached.

Your task is to design the software system and produce a detailed design document. Once that is ready, you are to implement the system in C on a UNIX platform (e.g. Linux or Mac OS X) and test the performance using the test harness. If you change your design during implementation or testing, you should update the design document.

You link the library provided to you with your code, calling appropriate functions (see API) from your main program. You are required to demonstrate the efficacy of your software by collecting statistics over a 5 minute test (300 apples). When the test is completed, the library displays some performance statistics.

Your code should be in a file called `ca1.c`, and you link it with the test framework with the following command:

```
gcc -o ca1 ca1.c libapples.a -lpthread -lm
```

The file `libapples.a` is provided via IVLE (as part of `libapples.tgz`). You may need to select the appropriate file for your machine architecture (various sub-folders in the `tgz` file[1]).

Submission

You are required to submit a `zip/tgz` file containing the following files:

`report.pdf` – a report describing your software design, not to exceed 3 pages
`ca1.c` – your implementation of the software described above
`output.txt` – the output from running your code, redirected to a file

The `zip/tgz` file should be named with your user-id (e.g. `g1234567.zip`) and uploaded to the "CA1 Submissions" folder on the IVLE workbin **by 12:00 pm (noon) on Febraury 16, 2016**.

Your report should be concise but complete, and should describe your software design and highlight its key features.

The `ca1.c` source code may be compiled and linked with the command shown in the previous section and the resulting executable `ca1` should generate similar output as you submit in `output.txt`.

Please follow the above guidelines exactly, as your program may be tested by an automated system.

[1] A `tgz` file is a compressed archive. You can extract all files from it on a UNIX command prompt using: `tar xvzf libapples.tgz`

Getting and giving help

If you have any questions about the assignment, please post them on the discussion forum in IVLE.

This is a learning exercise as well as an assessment. Since it is a learning exercise, you are encouraged to seek help, if needed, in order to complete the assignment. You are also encouraged to help others. The primary platform for seeking and giving help is the IVLE discussion forum so ensure that others in the class also have a chance to learn from the exchange.

Remember that you will not help others learn by giving them the solution – so please limit your help to pointers, advise and at most, example code snippets that do not directly form a substantial part of the solution. **Do not post or give any part of your solution to other students!**

If all else fails and you need personal consultation to complete the assignment, please email me to arrange for an appointment at least 1 week before the submission date. In order for me to be able to help you, you must first try to solve the problem on your own!

API Documentation

```
void start_test()
```

Starts a simulated test using the test harness. A test consists of up to 300 apples with a similar good/bad distribution as measured during the last sampling exercise. This function should be called before any other function in this API.

—

```
int more_apples()
```

Checks if more apples are available in the current test. Returns 1 if more apples are available, or 0 if all 300 apples have been packed or discarded.

—

```
void end_test()
```

Stops the current test and prints performance statistics. No other functions in this API should be called on the test has ended.

—

```
void wait_until_apple_under_camera()
```

Waits until the an apple on the conveyer belt is aligned within ±1 cm of the location of the camera.

—

```
PHOTO take_photo()
```

Takes a photograph of the apple under the camera. To get a good photograph, the apple must be aligned within ±1 cm of the location of the camera. Returns a handle to the photograph.

—

```
QUALITY process_photo(PHOTO photo)
```

Processes a photograph to determine if the apple in the photograph is good or bad. The photograph must be previously taken using `take_photo()` when the apple is properly aligned with the camera. If the photograph does not contain an apple (or the apple is poorly aligned with the camera), this function returns `UNKNOWN` as the quality. If the apple can be clearly identified, this function returns `GOOD` or `BAD`.

The image processing takes a variable amount of time to complete. The service time is approximately exponentially distributed with a mean of 990 ms. Only one photo can be processed at a time by the image processing unit, with additional calls to this function blocking if the unit is busy.

—

```
void discard_apple()
```

Discards the apple passing the actuator location. The apple must be within ±2 cm of the actuator location for it to be discarded correctly.

—