Robonaut challenge (part 1)

Sébastien Drouyer (handle: sdrdis)
1190 registrants, 40 competitors

I. Introduction

The Robonaut Challenge is part of the ISS Challenge Series organised by NASA in order to solve some issues related to the International Space Station. This challenge series is separated into three main challenges:

- The Longeron challenge, which goal was to optimize the power production of the ISS when placed on difficult orbital positions.
- The ISS-FIT challenge, which purpose was to create an iPad Application for monitoring the astronauts food intake.
- And the Robonaut Challenge, which can be separated into two smaller challenges :
    - The first one is to teach Robonaut 2 (an humanoïd robot created by Nasa) how to recognize state and location of several buttons and switches on a taskboard.
    - The second one is the use this vision algorithm to operate on this taskboard.

This document is about the first part of the Robonaut Challenge. Its purpose is to provide an overview of the solution as well as side tools that have been used.

II. Problem overview

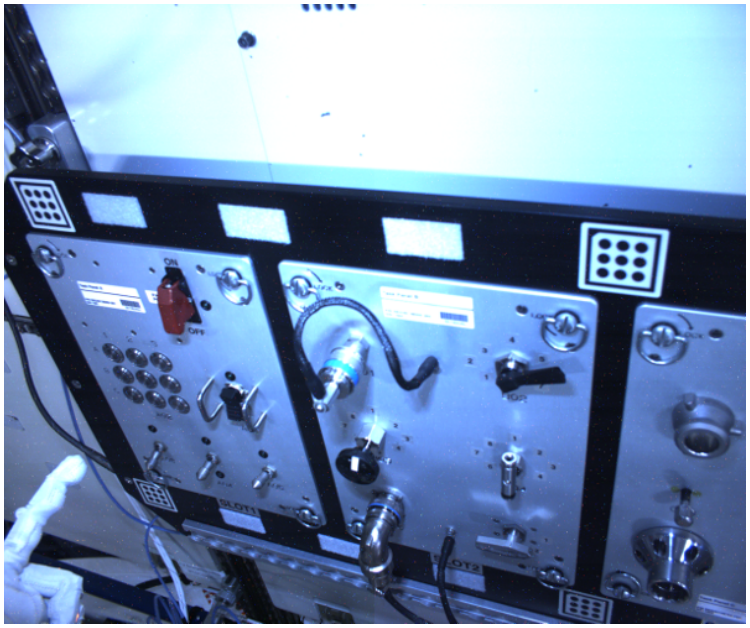
Robonaut 2 is the first humanoid robot in space and it was sent to the space station with the intention of taking over tasks too dangerous or too mundane for astronauts. But Robonaut 2 needs to learn how to interact with the types of input devices the astronauts use on the space station. And it has to recognize the current state of the taskboards before it can operate on them ; which buttons are already pushed, which leds are active, is the power switch flipped, and so on.

This is the purpose of the challenge. In order to do so, a several pictures of a unique taskboard is provided. These pictures are separated in 5 subsets :

- ISS: real pictures of the taskboard taken on the International Space Station.
- Lab, Lab2 and Lab3: real pictures of the taskboard taken on earth on different positions. Lab pictures are more are similar to the ISS pictures than Lab3.
- Sim: images generated by a 3d simulation of the taskboard.

A strong performance on the real imagery is preferable over simulated imagery. Therefore, the score coefficient of the ISS pictures are more important than the Lab pictures, which is more important than the Lab2, and so on.

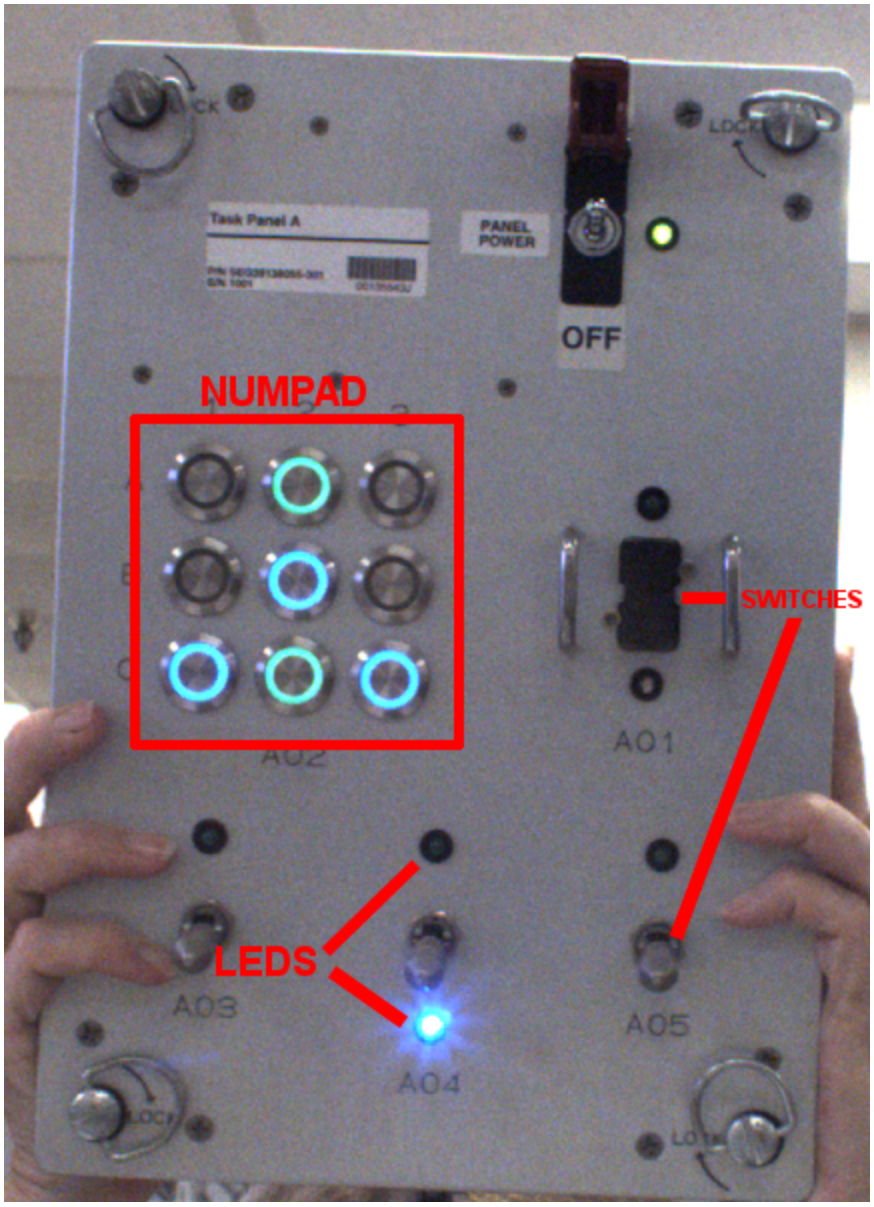
Simulated image of the taskboard


Real picture taken on the ISS

On the taskboard, the following items have to be recognized :

- Led positions and states (ON/OFF).

- Numpad buttons positions and state (ON/OFF).
- Switches positions and states state (UP/DOWN and CENTER for some)



Illustrations of numpad buttons, leds and switches on a Lab3 picture

III. Solution overview

A. General description

The solution provided is generalist, in the sense that the same process with the same settings are used for all subsets of pictures (from ISS to Sim).

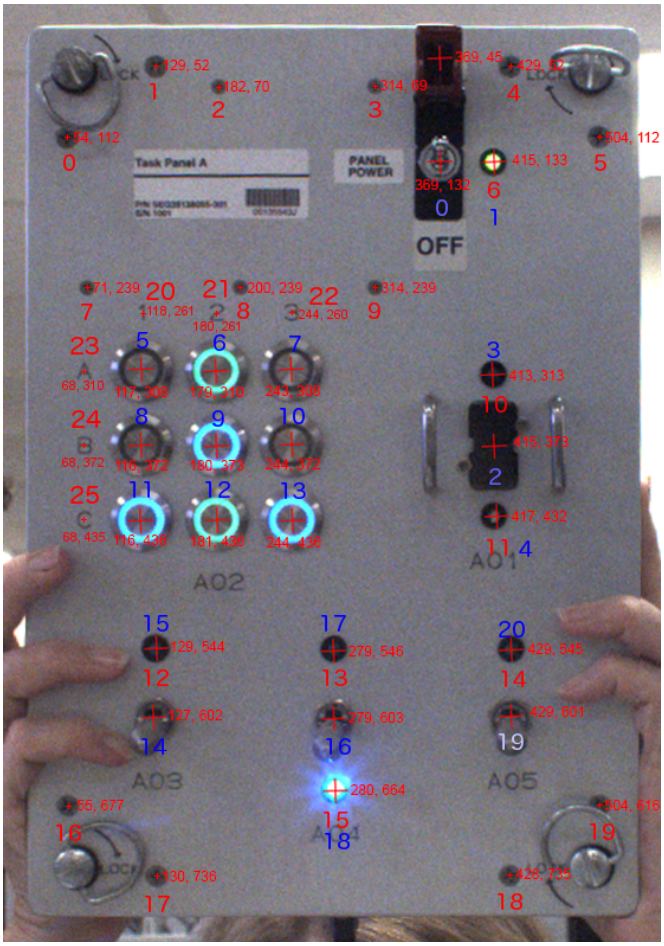
1. Secondary objects labelling

The first step of the analysis was to label “Secondary objects”. Secondary objects are small and easily detectables items in the image when using image segmentation. They do not have to be the items we need to detect (the “Primary objects”), but the idea is that they will contribute to localize them.

A Lab3 image have been selected and slightly transformed in order to simulate the taskboard as it would be exactly in front of the camera (a similar image was provided by NASA, but it was one day before the end of the extended contest). From there, some easily identifiable components such as screws, leds and numpad letters and numbers (A-B-C and 1-2-3), are labelled in red in the picture below.

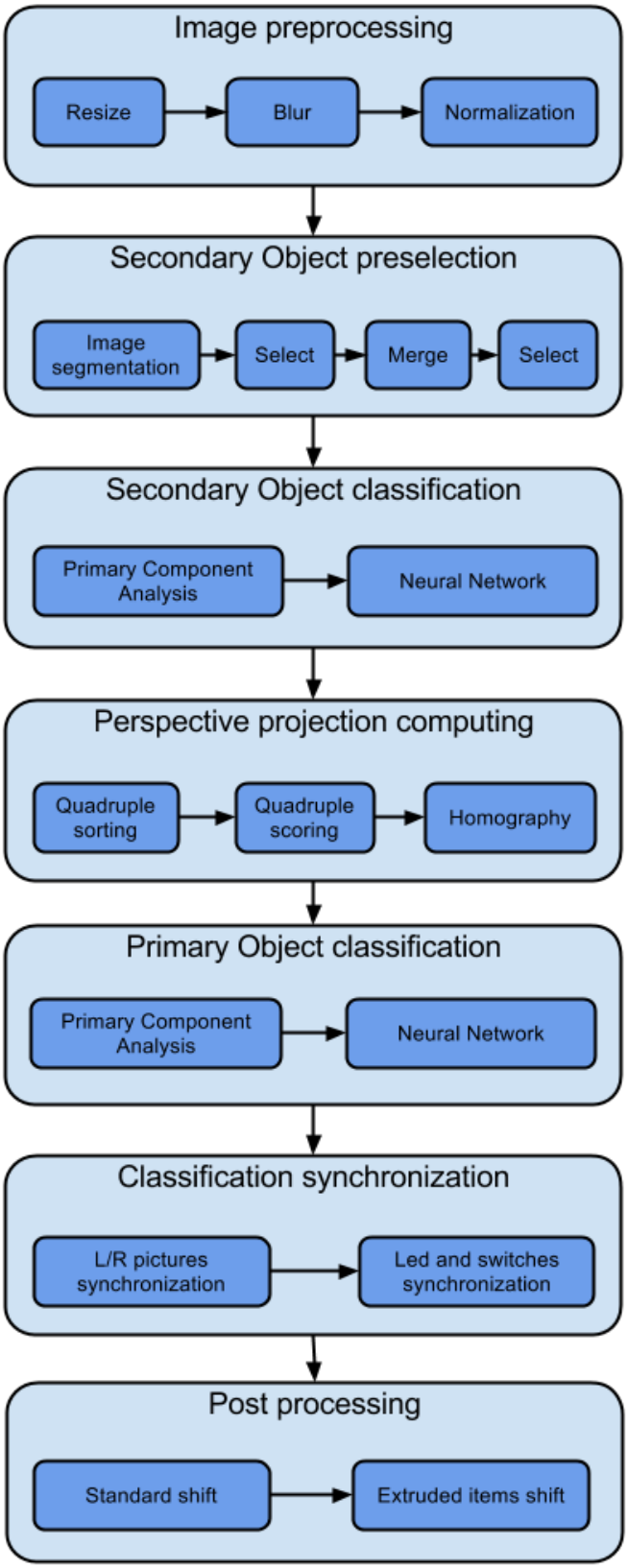
Primary objects have been labelled in blue. Note that the panel power cover is not considered as a primary object since is position is determined from the panel power switch : the consequence is that there is a shift in the expected identifiers other than 0.

There is 26 secondary objects and 21 primary objects on the taskboard. Note that some secondary objects can also be primary objects (leds).



Taskboard primary and secondary objects

2. General process



The image are processed following 7 main steps. The main idea is described below:

1. **Image preprocessing:** this step is necessary for an efficient image segmentation. First, a smaller copy of the image is created, on which we apply a blur filter. Then the image is normalized since some images are darker / brighter and in order to enhance the image contrast.
2. **Secondary object preselection:** on this step the objective is to identify small objects that **could** be secondary object. We segment the smaller image using the “Efficient Graph-Based Image Segmentation” algorithm by Pedro F. Felzenszwalb and Daniel P. Huttenlocher. We then select small and consistent elements, merge those small elements when they share a high amount of threshold, and select again.
3. **Secondary object classification:** once these small elements have been preselected, we use trained Primary Component Analysis and Neural Network in order to determine whether or not those elements are secondary objects, and if that is the case the class of the secondary object : SCREW, LED, or NUMPAD letter / number.
4. **Perspective projection computing:** if we have at least 4 secondary objects, we can then try compute the perspective projection of the device. To do that, first we generate a list of secondary objects quadruples ordered from smallest to the biggest. For each quadruple of the list we then formulate hypotheses about the actual position they are on the device, and calculate an homography matrix. We score these hypotheses according to the number of detected secondary objects the calculated homography projection match. We select then the best hypothesis to calculate the final homography matrix using all matched points.
5. **Primary object classification:** The final homography matrix allows us to know primary object positions. All we have to do now is to determine the state of the primary objects : OFF / ON, DOWN / CENTER / UP. Once again, we do that using trained Primary Component Analysis and Neural Network.
6. **Classification synchronization:** We have been given two pictures - left and right - and sometimes the states detected for the same primary object on the two image can be different. They are also dependencies between primary object states ; for example when a switch is up there is always an associated led on. So this step consists to synchronize the states in order to return a viable hypothesis.
7. **Post processing:** There is often a shift in the detected primary object position, so they are moved by few pixels. There are also some extruded items (as the panel power cover and the rocker switch) which position can’t be determined using homography. Their position is determined using linear regression.

B. Requirements

The two images (left and right) have to be in similar conditions than the test images provided : they can’t be too near nor too far, lighting conditions and orientation must comparable. Since we use homography, at least 4 valid secondary objects have to be visible, but this is not likely to be sufficient (more than 8 valid secondary objects is optimum). These requirements exist only for performance enhancement so it is possible to loosen them, though it might have a strong impact on runtime.

C. Technologies used

The submission was written in C++, using the TNT framework (for matrices manipulation, svd and homography) and the "Efficient Graph-Based Image Segmentation" by Pedro F. Felzenszwalb and Daniel P. Huttenlocher.

However, the submission is only a minor part of the main program developed for this contest. The main program use the Qt framework for the image and file classes (which allows us to easily load / manipulate / save images and files) and the convenient interface forms. Interface forms have been designed for a 1920x1080 screen, although changing the form size should be easy with the Qt Creator (the Qt IDE) interface.

A small utility has been implemented using PHP (generate_bundle.php) which allow us to extract the submission code into a single bundle we can easily copy in the TopCoder interface. It is included since it was useful for the contest, although we are not sure it will of any use to NASA.

PCAs and Neural Networks were trained using Octave. It is a derived version of the exercise scripts provided in the coursera Machine Learning courses by Andrew Ng. Small modifications were made for standardization as well as to allow multiprocessing. Data export from Octave to C++ are done using an Octave script which has been implemented for this contest.

IV. Solution detailed description

We will explained here the solution more in detail and show how parameters were chosen.

A. Image preprocessing and Secondary object preselection

The two first steps are explained together since they are dependent on each other. Parameters were obtained manually, so there is a place for improvement here.

1. Image resize

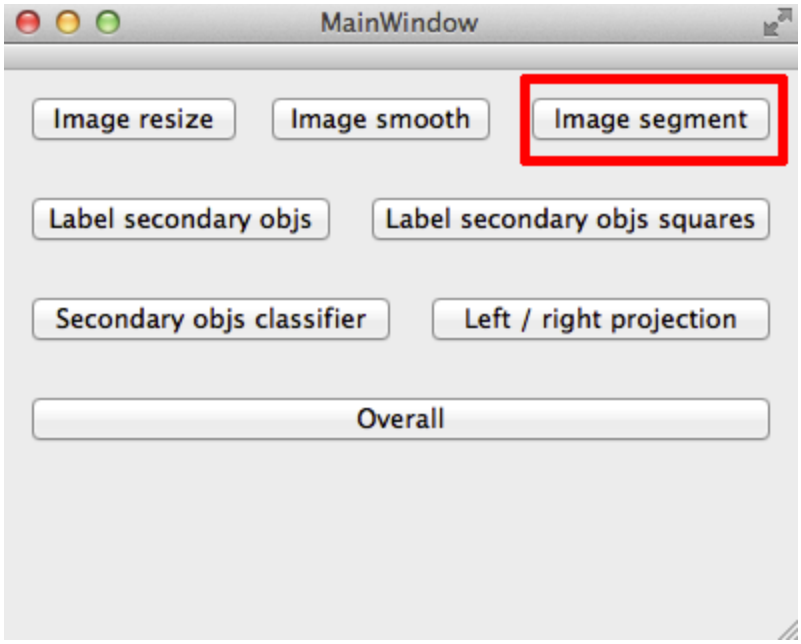
The first step is to get a resized image of the original one (same ratio). This must be done knowing:

- Reducing the image size will allow greater performance on the image segmentation phase.
- Secondary objects on the resized image must remain visible and differentiable so they can be isolated during the segmentation step.

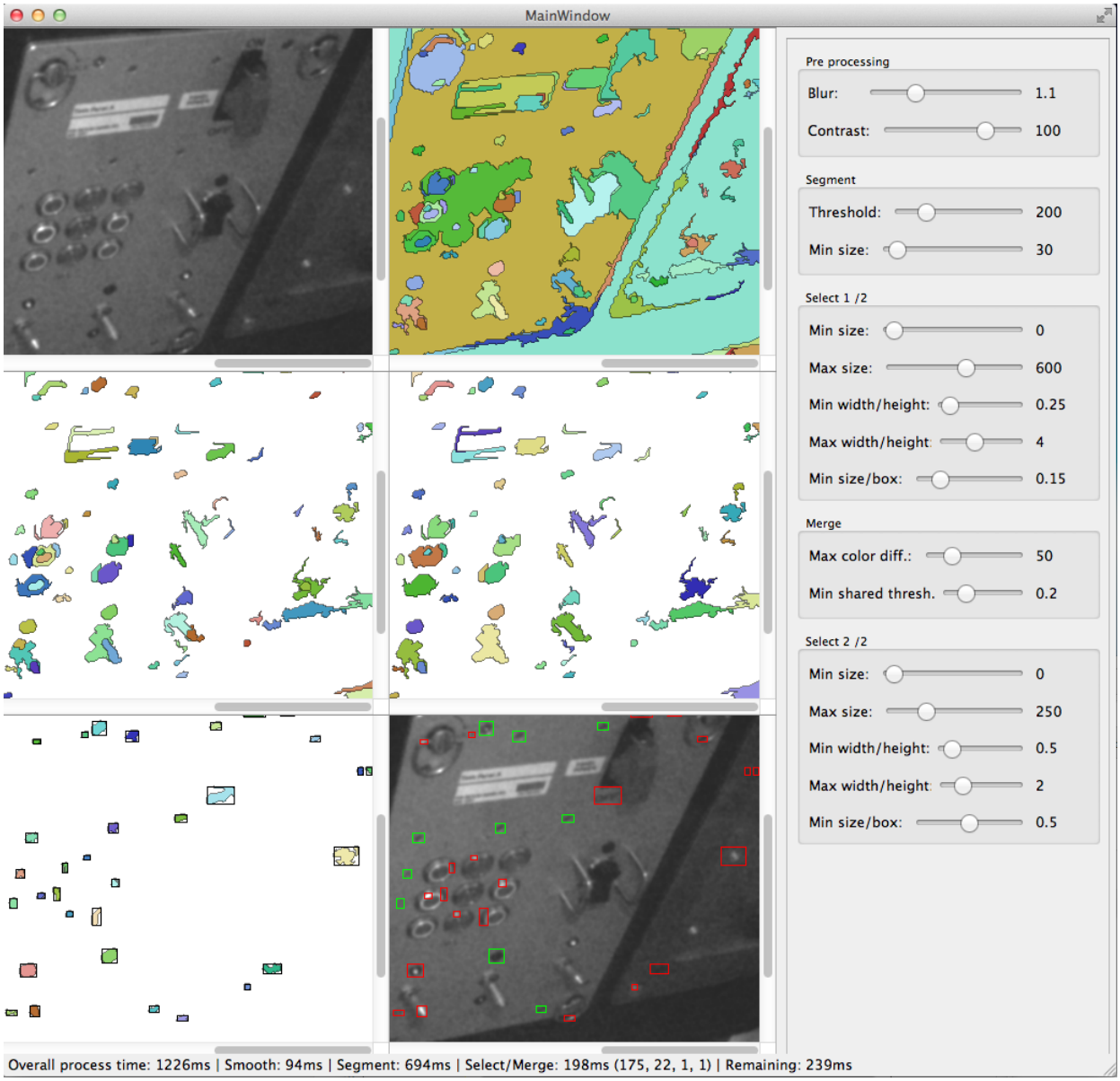
We have chosen than the resized image would be 800px width. This was done using some general manual observation.

2. The Image segment window

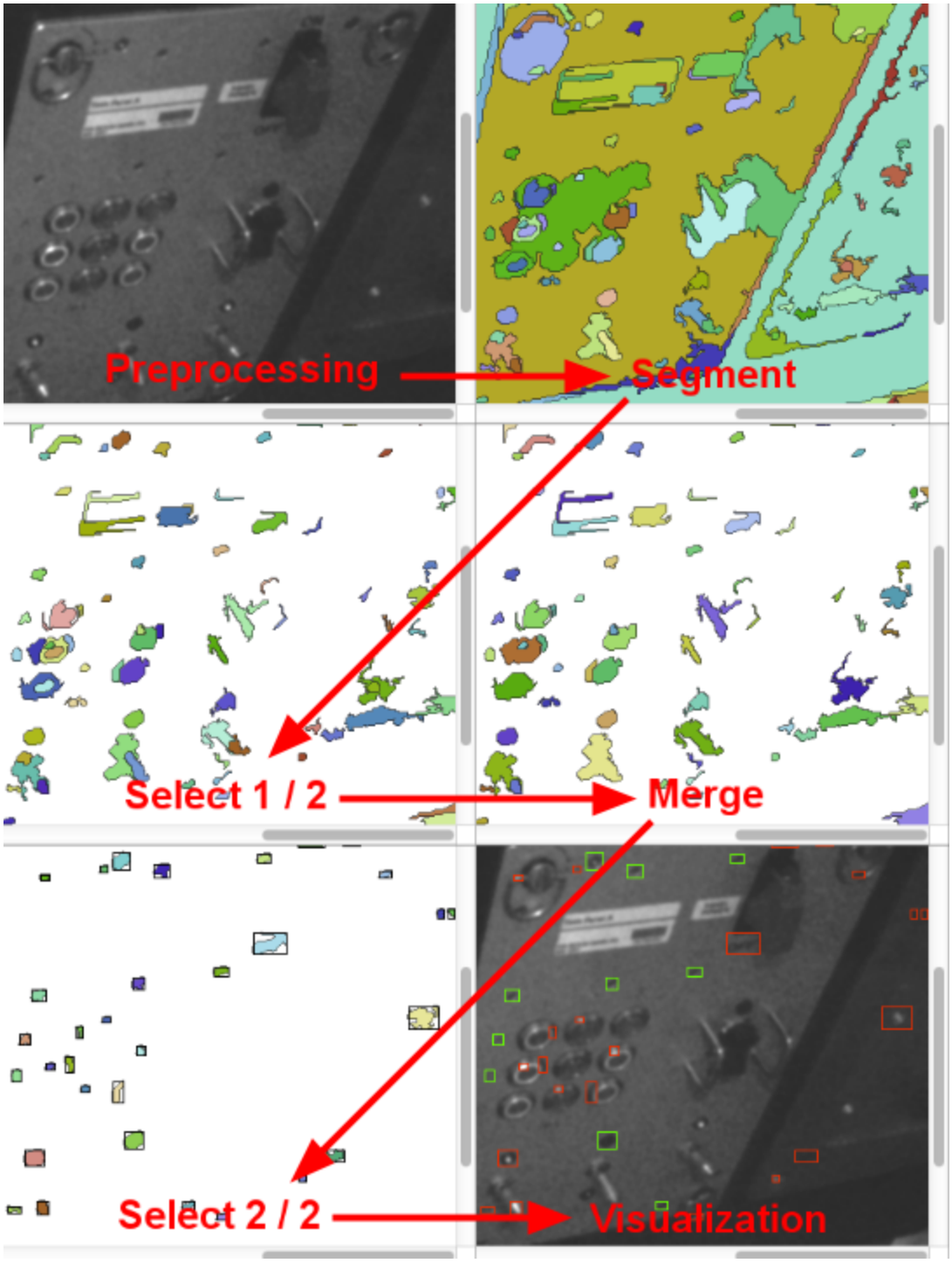
The parameters in the next steps have been chosen with an form developed in Qt.



On the starting window, the window can accessed through the “Image segment” button



Overview of the Image segment window



How to understand the “Image segment” window

The purpose of this window is to choose each step parameters manually by visualizing the results in real time. While the chosen parameters are probably not optimum in the global scope, it allows us to get an idea of the effect (or not) a parameter have on the process.

We can see on the window that there are 5 intermediary steps.

a. Preprocessing

Blur (chosen value: 1.1): Blur filter applied on the resized image
Contrast (chosen value: 100): The image is normalized. We then change the mean to 128 and sigma to the chosen contrast parameter.

b. Segment

We use here the two parameters of the “Efficient Graph-Based Image Segmentation” by Pedro F. Felzenszwalb and Daniel P. Huttenlocher.

Threshold (chosen value: 200): Maximum intensity difference allowed between each segmentation area.
Min size (chosen value: 30): Minimum size of each segmentation area.

c. Select 1 / 2

This step allows us to perform a selection on segmentation areas.
Here are 3 properties we know about secondary objects :

- They are small
- Their width / height ratio is generally near 1.
- They are dense, compact.

This selection is therefore made upon 5 parameters :

Min size (chosen value: 0): The minimum size of the area (number of points).
Max size (chosen value: 600): The maximum size of the area (number of points).
Min width / height (chosen value: 0.25): The minimum width over height ratio.
Max width / height (chosen value: 4): The maximum width over height ratio.
Min size / box (chosen value: 0.15): The minimum ratio of the area size (number of points) over the area of the bounding box. This ratio is interesting since if the area is a compact rectangle its value will be near 1, and if the area is very sparse, its value will be near 0.

d. Merge

This step allows us to merge close segmentation areas. The idea is that sometimes a secondary object have a brighter and a darker side which will be separated into 2 different areas which we need to merge.

Max color diff (chosen value: 50): Max color difference between two areas to be merged.
Min shared thresh. (chosen value: 0.2): Minimum amount of shared threshold between two areas to be merged. A value of 0.2, it means that two areas need to have at least 20% of shared

threshold.

e. Select 2 / 2

This step is similar to the first one except it occurs after the merge. Chosen parameters are more selective.

Min size (chosen value: 0)
Max size (chosen value: 250)
Min width / height (chosen value: 0.5)
Max width / height (chosen value: 2)
Min size / box (chosen value: 0.5)

3. Preprocessing before classification

Once the preselection has been done, we extract the image of the segmentation areas bounding squares resized to 20x20.

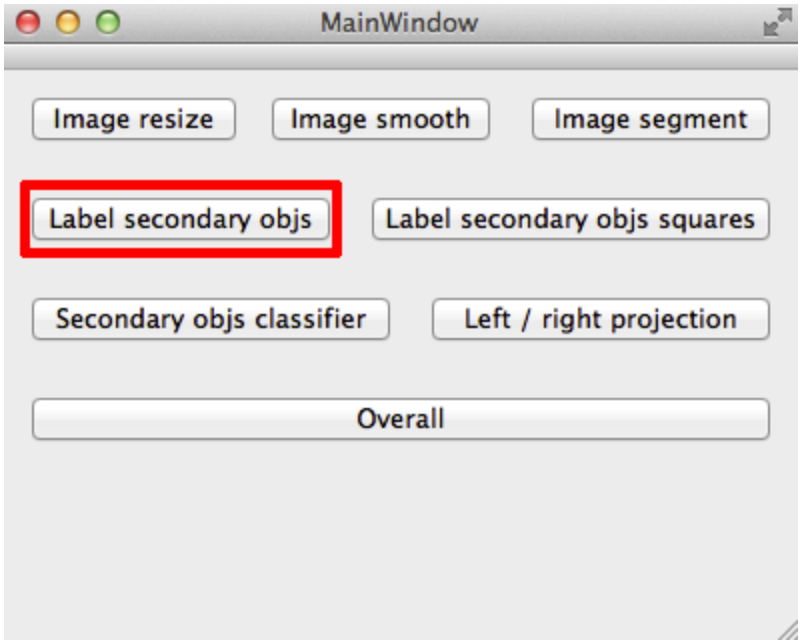
B. Secondary object classification

The first step combined to the second step gives us a number of segmentation areas which size and shape are similar to secondary object in general. The objective of this step is to determine whether or not our areas are secondary objects and if that is the case the class of the secondary object: **SCREW**, **LED**, or **NUMPAD** letter / number.

1. Secondary object labelling

In order to train PCA and Neural Network, we needed a training set. This training set has been generated manually using two windows.

a. Secondary object localisation window



On the starting window, this window can be accessed through the “Label secondary objs” button

Overview of the “Label secondary objs” window

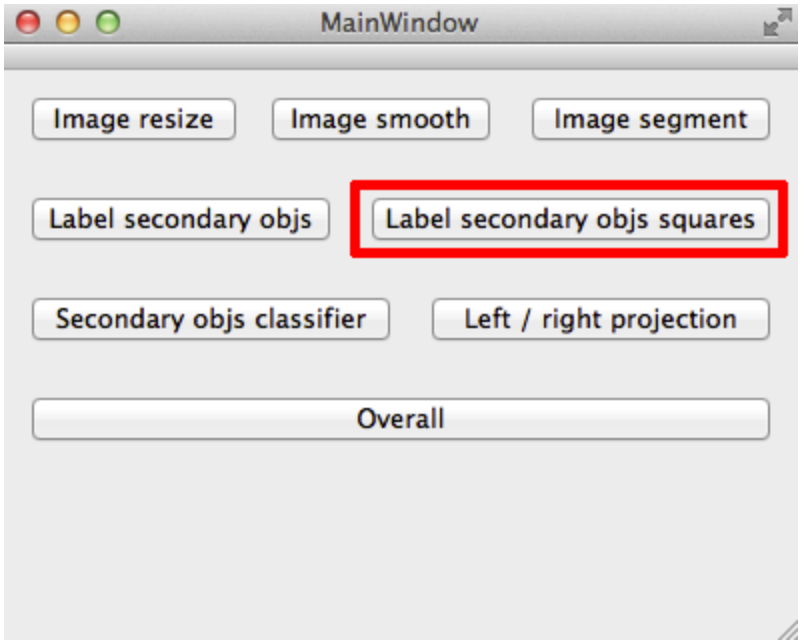
Using the first window, we localise each secondary object which are not already primary object (the leds) since we already know their position. On the screenshot above, it is possible to observe crosses : green crosses are primary objects, red cross are screws and yellow cross are numpad letters / numbers.

To enter a new position, a simple left click on the image is needed. To remove a position, a right click will remove the nearest position. It is possible to switch image with the “Previous” and “Next” buttons, or with arrow keys.

Please note that since the purpose of this tool was to be used internally, some things are

hardcoded such as the type of position (SCREW or NUMPAD) to be added. But with few changes this problem should be easily solved.

b. Secondary object squares localisation window



On the starting window, this window can be accessed through the “Label secondary objs squares” button

Overview of the “Label secondary objs squares” window

The idea of this window is to combine the secondary object preselection step with the secondary object localisation data. We apply the first two steps in order to obtain possible secondary objects (segmentation areas which have the same shape and size).

For each area, we lookup if there are any marked secondary object. If there are, we consider the area as positive (green in the above picture), if not, negative (red in the above picture). It is possible to change manually this state by clicking near the area.

Please note we are not only labelling secondary objects. As you can see in the above screenshot, some external screws were also labelled positive since they have exactly the same

aspect as the one we are searching for.

This way, we will be able to export training pictures as near as possible as they will be when running the algorithm.

2. Training

As described earlier, we use octave to train PCA and Neural Network. This occurs in three phases.

a. Collect

The exported training pictures need first to be transformed to octave data. The input data is images of 20x20 so 400 numbers for each image. The output values are:

- 0: not a secondary object
- 1: screw
- 2: led
- 3: numpad

b. PCA

Then we visualize the result of PCA with different numbers of eigenvectors. Here we decided to use **80** eigenvectors (it was chosen manually). We then export these eigenvectors to valid C++ code.

c. Neural Network training

Then from the simplified dataset, taking advantage of multiprocessing, we trained multiple neural networks with different parameters for 100 iterations:

- Number of hidden layer: 70, 80, 90, 100, 110, 120, 130, 140, 150
- Lambda (regularization): 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30

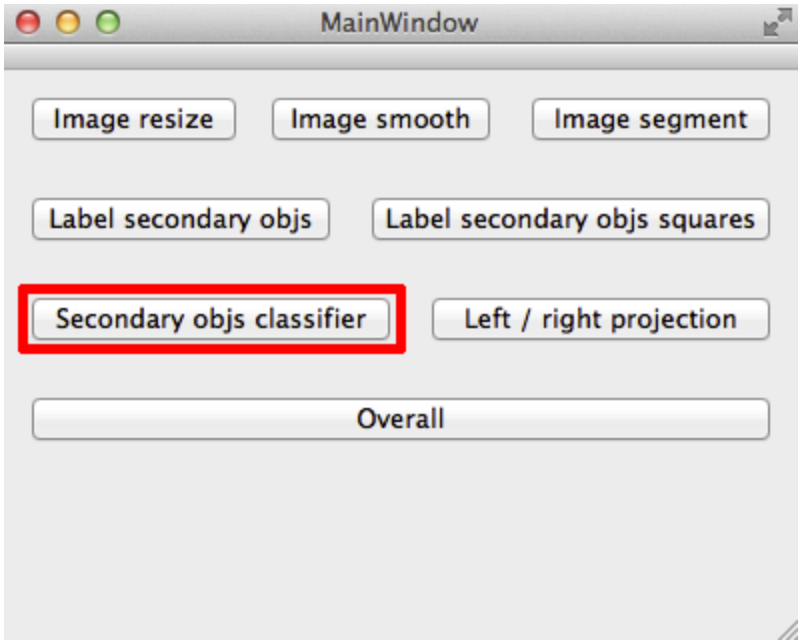
Once all was trained we selected the best neural network according to the F1 score over the 0 output value, therefore:

- Number of hidden layer: **80**
- Lambda: **0.01**
- With F1 score: **0.91**

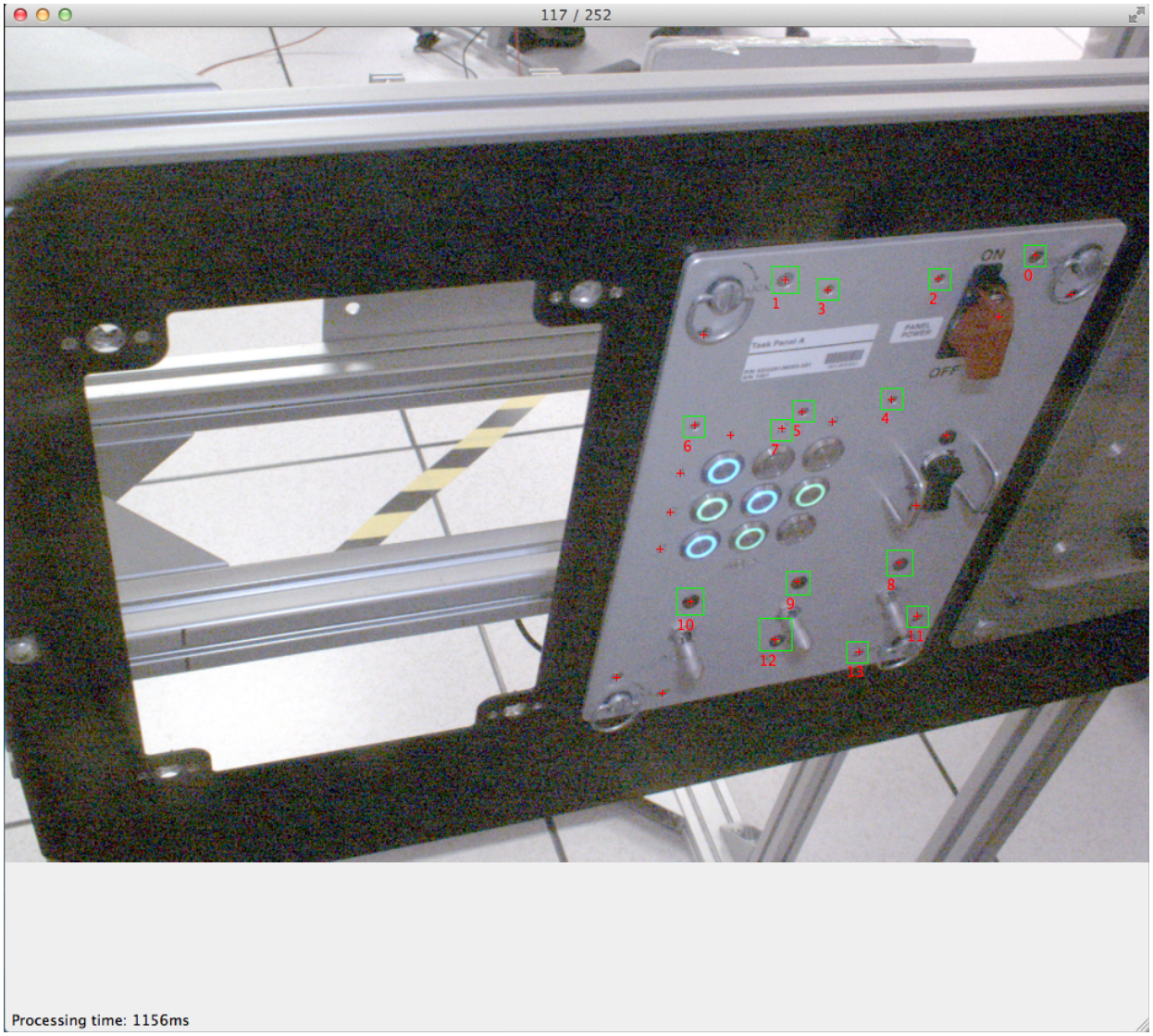
We then export the different weights to valid C++ code.

3. Visualization

A window has been designed in order to visualize the result.



On the starting window, this window can be accessed through the “Secondary objs classifier” button



Overview of the “Secondary objs classifier” window

This window shows the validated secondary objects. Image can be switched using keyboard arrows.

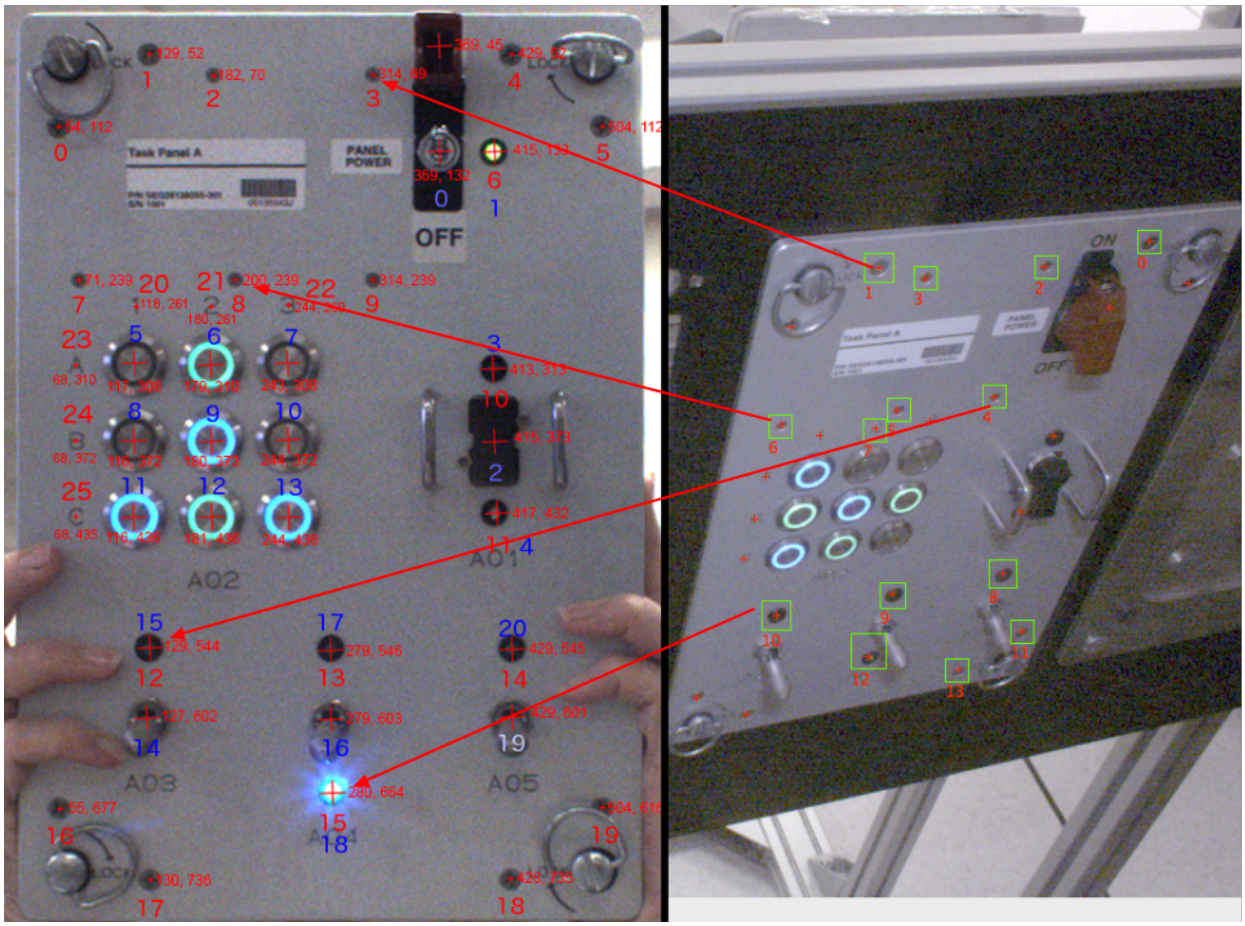
C. Perspective projection computing

To calculate a perspective projection of the device, we need at least 4 points. If we do, this step needs to address two issues:

- Validated secondary objects won’t necessarily appear always in the same order and there will be missing objects as well as false positives. We need to find the most satisfying set, knowing we will rarely be able to get a match for all detected secondary objects.
- Even from the most satisfying set, there can be some error in the matched secondary objects. For instance if a false positive is detected near an expected secondary object, this can have a strong negative impact on the perspective projection.

The idea in this step is to formulate a big number of hypothesis.

For each hypothesis, we select a quadruple of detected secondary objects and we suppose they match 4 real secondary objects.



Then from the resulting homography we calculate a score depending on how many detected secondary objects match real ones according to the projection. The best hypothesis is the one with the best score.

As the main idea is simple, there are serious performance issues that need to be addressed. Indeed:

- There are C(n, 4) possible quadruples, n being the number of detected secondary objects.
- For each quadruple, there are $21 \times 20 \times 19 \times 18 = 143640$ hypothesis (since there are 21 real secondary objects to be matched)
- For evaluating each hypothesis, we need to calculate at least one perspective projection, that is compute an homography using SVD and matrices inversions.

We will describe how this issues were solved in the two next parts.

1. Quadruple sorting

The first idea is that not all quadruples need to be evaluated:

- Contracted quadruples - composed of very near secondary objects - don't need to be evaluated since the resulting projection will lack of precision.
- If a quadruple is sparse - composed of very distant secondary objects - then it is probably composed of at least one false positive secondary object.

Therefore, quadruples are sorted ascendingly using the sum of the distances between each points and the quadruple center. If this sum is inferior to **250** then this quadruple is ignored.

2. Quadruple scoring

Then for each quadruple we need to evaluate the best hypothesis. A first approach is to do 4 simple loops (A, B, C, D are the quadruple points) over real secondary objects (see picture on previous page) :

```
for (i = 0; i < 21; i++) {
  for (j = 0; j < 21; j++) {
    if (i == j) continue;
    for (k = 0; k < 21; k++) {
      if (i == k || j == k) continue;
      for (l = 0; l < 21; l++) {
        if (i == l || j == l || k == l) continue;
        score = evaluateHypothesis(A, B, C, D, i, j, h, l);
        // test if best score, if that is the case save it
      }
    }
  }
}
```

As we said earlier, the problem of this approach is that it tests 143640 hypothesis, which is a lot considering we have to do at least one perspective projection each time.

So the objective is to lower this number of hypothesis.

a. Class testing

First, we test if the class of the detected secondary object (**SCREW**, **LED**, or **NUMPAD**) is the same than the one we want to match.

To be more precise, the neural network return 4 scores between 0 and 1 related to the probability than the secondary object is from one class (**NEGATIVE**, **SCREW**, **LED**, or **NUMPAD**). We consider that a detected object is from a defined class if this score is superior to **0.35** (meaning we slightly prefer high recall than high precision).

```
for (i = 0; i < 21; i++) {
  if (!isSameClass(A, i)) continue;
```

```
  for (j = 0; j < 21; j++) {
    if (i == j) continue;
    if (!isSameClass(B, j)) continue;
    for (k = 0; k < 21; k++) {
      if (i == k || j == k) continue;
      if (!isSameClass(C, k)) continue;
      for (l = 0; l < 21; l++) {
        if (i == l || j == l || k == l) continue;
        if (!isSameClass(D, l)) continue;
        score = evaluateHypothesis(A, B, C, D, i, j, h, l);
        // test if best score, if that is the case save it
      }
    }
  }
}
```

b. Orthographic heuristic

At this stage we are still testing for all permutations, although we know:

- The device is nor too near nor too far
- There are bounds to the orientation (the device cannot be upside down).

Therefore, we could for instance test that the matching secondary object 1 is higher than the matching secondary object 17.

A 2.0 version of this first idea is to calculate orthographic projection of the 4 sets of 3 secondary objects. From this orthographic projection - $X = ax + by + c$, $Y = a'x + b'y + c'$ - we can get:

- The equation parameters (a, b, c, a', b', c')
- The projected vector (1, 0) distance and angle

From statistic observations over our manually built dataset (you can find these statistics on the orthographic_stats.ods file), we can apply some boundaries on these parameters:

- 0.5 < vector distance < 3.25
- -1 < vector angle < 1
- 0.25 < a < 2.5
- -0.5 < b < 0.75
- 0.25 < a' < 2
- -1.5 < b' < 2

If the orthographic projection of any of the 4 sets doesn't respect all boundaries, then it is considered incorrect, and the hypothesis skipped. Each boundary was designed to have around 98% recall.

If the 4 sets are considered correct, we compare their parameters because they have to be similar if the hypothesis is correct. For each parameters, we calculate standard deviation

between the 4 sets. Here are boundaries deduced from statistics:

- std(vector distance) < 1.4
- std(vector angle) < 0.8
- std(a) < 1.4
- std(b) < 0.8
- std(c) < 350
- std(a') < 0.8
- std(b') < 2.2
- std(c') < 750

Each boundary was designed to have around 97% recall.

```
for (i = 0; i < 21; i++) {
  if (!isSameClass(A, i)) continue;
  for (j = 0; j < 21; j++) {
    if (i == j) continue;
    if (!isSameClass(B, j)) continue;
    for (k = 0; k < 21; k++) {
      if (i == k || j == k) continue;
      if (!isSameClass(C, k)) continue;
      if (!seemCorrect(A, B, C, i, j, k)) continue;
      for (l = 0; l < 21; l++) {
        if (i == l || j == l || k == l) continue;
        if (!isSameClass(D, l)) continue;
        if (!seemCorrect(A, B, D, i, j, l)) continue;
        if (!seemCorrect(A, C, D, i, k, l)) continue;
        if (!seemCorrect(B, C, D, j, k, l)) continue;
        if (!seemCorrect(A, B, C, D, i, j, k, l)) continue;
        score = evaluateHypothesis(A, B, C, D, i, j, h, l);
        // test if best score, if that is the case save it
      }
    }
  }
}
```

c. Premature exit

When the hypothesis score is over a lower bound, we can then exit the algorithm because we are virtually sure the hypothesis is correct.

First we must explain how this score is calculated : from the perspective projection, we search for each projected point the nearest detected secondary object (with distance < 30). If the distance is 0 then we add 100 to the score, if the distance is superior to 30 then nothing is added to the score. In between $((30 - distance)/30)^2 \times 100$ is added to the score.

The lower bound score is $max(550, min(850, 0.6 \times n \times 100))$, n being the number of detected secondary objects.

3. Homography

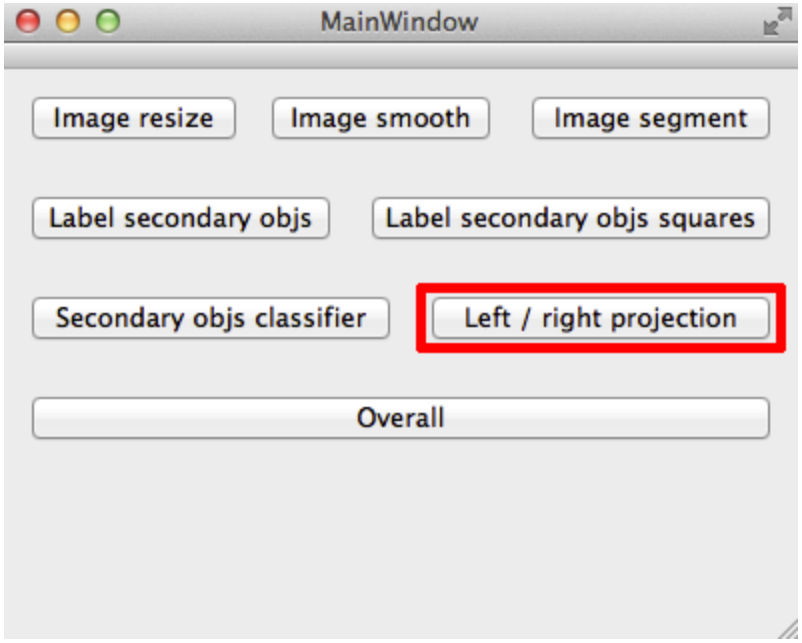
Since we calculate homography from SVD, it is very sensitive to any false positive.

We therefore implemented the concept of purified homography, where we try, if possible, to remove one or two matches and see if the score gets improved. This method is called when scoring hypothesis and at the end once the best hypothesis have been selected.

This makes the hypothesis evaluation heavier but more precise allowing us to limit the number of hypothesis evaluated.

4. Visualization

In order to visualize this step result (+ primary object classification), a Qt form has been implemented.



On the starting window, this window can be accessed through the “Left / right projection” button

Window overview

Images can be switched using the keyboard arrow keys. The overview is combined with the next step, but it is possible to observe the device boundaries in white, deduced from homography.

D. Primary object classification

Once the homography matrix is calculated, we can then deduce the primary objects positions.

In the next step, we must then classify each primary object. To do so, we define a bounding box around the deduced position, which size depends on perspective projection.

We then resize this bounding box to 30x30 pixels. This process is used for the running algorithm as well as the generation of the training set (we manually remove results when homography is incorrect).

1. Training

We use here the same overall process as for secondary objects.

a. Collect

The exported training pictures need first to be transformed to octave data. The input data is images of 30x30 so 900 numbers for each image. The output values are:

- 0: DOWN / OFF
- 1: UP / ON
- 2: CENTER

b. PCA

Then we visualize the result of PCA with different numbers of eigenvectors. Here we decided to use **100** eigenvectors (it was chosen manually). We then export these eigenvectors to valid C++ code.

c. Neural Network training

Then from the simplified dataset, taking advantage of multiprocessing, we trained multiple neural networks with different parameters for 100 iterations:

- Number of hidden layer: 70, 80, 90, 100, 110, 120, 130, 140, 150
- Lambda (regularization): 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30

Once all was trained we selected the best neural network according to the F1 score over the 2 output value, therefore:

- Number of hidden layer: **170**
- Lambda: **0.03**
- With F1 score: **0.96**

We then export the different weights to valid C++ code.

2. Visualization

The result can be overview with the “Left / right projection” window we show earlier.

Window overview

We can clearly see bounding boxes here. The stroke color depends on the detected state :

- Red if DOWN / OFF
- Green if UP / ON
- Yellow if CENTER

Pictures can be switched using keyboard arrows.

E. Classification synchronization

This step is relatively simple.

1. Left / right pictures synchronization

For each primary object, the neural network returns 3 numbers (one for each state / class) between 0 and 1 related to the probability. To get an overall over two pictures, we simply add these probabilities.

2. Led and switch synchronization

Led and switches are related : for instance, if A03_TOGGLE is UP then the associated A03_LED is ON. To get an overall state, we sum up probabilities as in the left / right pictures. Except here there can be some weight ; for instance, since the A04_LED_BOTTOM is often hidden by the A04_TOGGLE, then its associated weight will be inferior to A04_LED_TOP. Here are the different weights:

- A01_ROCKER_SWITCH: 0.5
- A01_ROCKER_LED_TOP: 1
- A01_ROCKER_LED_BOTTOM: 0.2
- A03_TOGGLE: 0.5
- A03_LED: 1
- A04_TOGGLE: 0.5
- A04_LED_TOP: 1
- A04_LED_BOTTOM: 0.2
- A05_TOGGLE: 0.5
- A05_LED: 1

F. Post processing

1. Standard shift

Lot of position were a little upper than expected so we decided to add 5 pixel along the y axis on all non extruded primary objects.

2. Extruded items shift

The homography allows us to get almost all positions except those which are extruded from the device : the A01_ROCKER_SWITCH and the PANEL_POWER_COVER (in two states here).

A01_ROCKER_SWITCH has already an associated primary object. For PANEL_POWER_COVER, we take PANEL_POWER_SWITCH primary object as reference. From these positions, we project two vectors (0, 1) and (1, 0) and get their lengths and angles (length1, length2, angle1, angle2).

Although we don't have any perfect solution, we decided to perform a linear regression from these data to get the final positions.

a. A01_ROCKER_SWITCH

Below are parameters over which we applied linear regression:

- 1
- cos(angle1)
- sin(angle1)
- cos(angle2)
- sin(angle2)
- length1
- length1 * cos(angle1)

- length1 * sin(angle1)
- length1 * cos(angle2)
- length1 * sin(angle2)
- length2
- length2 * cos(angle1)
- length2 * sin(angle1)
- length2 * cos(angle2)
- length2 * sin(angle2)

On validation set, around 90% was less than 10 pixels away than expected position.

b. PANEL_POWER_COVER, state DOWN

Below are parameters over which we applied linear regression:

- 1
- sin(angle1)
- cos(angle2)
- sin(angle2)
- length1
- length1 * sin(angle1)
- length1 * cos(angle2)
- length1 * sin(angle2)

On validation set, around 97% was less than 10 pixels away than expected position.

c. PANEL_POWER_COVER, state UP

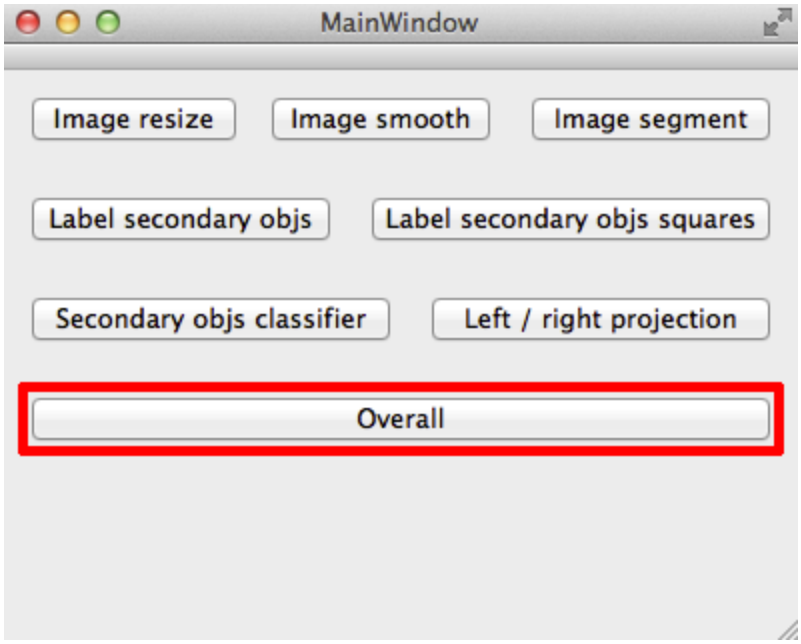
Below are parameters over which we applied linear regression:

- 1
- cos(angle1)
- sin(angle1)
- cos(angle2)
- sin(angle2)
- length1
- length1 * cos(angle1)
- length1 * sin(angle1)
- length1 * cos(angle2)
- length1 * sin(angle2)
- length2
- length2 * cos(angle1)
- length2 * sin(angle1)
- length2 * cos(angle2)
- length2 * sin(angle2)

On validation set, around 50% was less than 10 pixels away than expected position, and 60% less than 15 pixels aways. This is not a very surprising result since PANEL_POWER_COVER is very extruded when its state is UP.

G. Overall overview

A window has been implemented in order to have an overview of the final result.



On the starting window, this window can be accessed through the “Overall” button

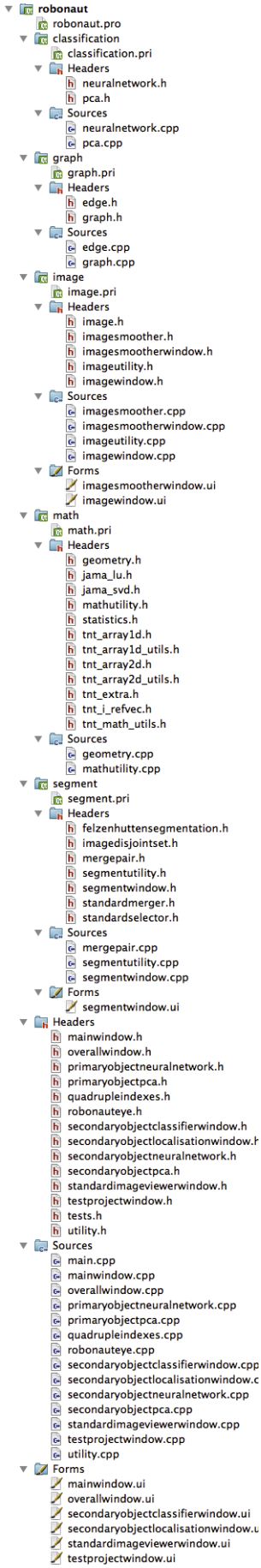
Overview of the “Overall” window

Pictures can be switch using keyboard arrows.

IV. Overview of the implementation

As explained before, there are two main parts : the main program in C++ / Qt, and the training scripts which are using Octave.

A. The main program


robonaut
robonaut.pro
classification
classification.pri
Headers
neuralnetwork.h
pca.h
Sources
neuralnetwork.cpp
pca.cpp
graph
graph.pri
Headers
edge.h
graph.h
Sources
edge.cpp
graph.cpp
image
image.pri
Headers
image.h
imagesmoother.h
imagesmootherwindow.h
imageutility.h
imagewindow.h
Sources
imagesmoother.cpp
imagesmootherwindow.cpp
imageutility.cpp
imagewindow.cpp
Forms
imagesmootherwindow.ui
imagewindow.ui
math
math.pri
Headers
geometry.h
jama_lu.h
jama_svd.h
mathutility.h
statistics.h
tnt_array1d.h
tnt_array1d_utils.h
tnt_array2d.h
tnt_array2d_utils.h
tnt_extra.h
tnt_i_refvec.h
tnt_math_utils.h
Sources
geometry.cpp
mathutility.cpp
segment
segment.pri
Headers
felzenhuttensegmentation.h
imagedisjointset.h
mergepair.h
segmentutility.h
segmentwindow.h
standardmerger.h
standardselector.h
Sources
mergepair.cpp
segmentutility.cpp
segmentwindow.cpp
Forms
segmentwindow.ui
Headers
mainwindow.h
overallwindow.h
primaryobjectneuralnetwork.h
primaryobjectpca.h
quadrupleindexes.h
robonauteye.h
secondaryobjectclassifierwindow.h
secondaryobjectneuralnetwork.h
secondaryobjectpca.h
standardimageviewerwindow.h
testprojectwindow.h
tests.h
utility.h
Sources
main.cpp
mainwindow.cpp
overallwindow.cpp
primaryobjectneuralnetwork.cpp
primaryobjectpca.cpp
quadrupleindexes.cpp
robonauteye.cpp
secondaryobjectneuralnetwork.cpp
secondaryobjectpca.cpp
standardimageviewerwindow.cpp
testprojectwindow.cpp
utility.cpp
Forms
mainwindow.ui
overallwindow.ui
secondaryobjectclassifierwindow.ui
standardimageviewerwindow.ui
testprojectwindow.ui

The main program has been designed for reusability. The root folder contains code which are specific to this problem, but there are 5 folders which contain classes for general use:

- **classification**: this folder includes the NeuralNetwork and PCA classes needed for classification. This classes are then extended in the main program (see primaryobjectneuralnetwork.cpp or primaryobjectpca.cpp for instance).
- **graph**: this folder includes a rough outline of classes as Graph and Edge. Since they have been implemented for the image segmentation routine they are not complete at all.
- **image**: this folder includes the Image class as well as ImageSmoother which allows us to apply a blur filter.
- **math**: this folder includes the TNT library which allows us to do matrices manipulations, and the JAMA library which has been added to compute SVD. It also include some custom classes, as Statistics (mean, variance, normalize, lower / upper bounds calculations), and Geometry (orthographic and perspective projection computing).
- **segment**: this folder includes the “Efficient Graph-Based Image Segmentation” algorithm by Pedro F. Felzenszwalb and Daniel P. Huttenlocher, (the class has been named FelzenHuttenSegmentation for a better comprehension), as well as the StandardMerger and StandardSelector classes which allows us to apply the Merge and Select steps as described before.

The central algorithm can be found in the robonauteye.cpp / robonauteye.h files. **Utility** is a convenient class, as it contains all methods that allowed us to export secondary object / primary object test cases.

To be easily used, the executable needs a data folder (take a look at cpp/robonaut-build-Desktop_Qt_5_0_1_clang_64bit-Debug). In this folder, there are manually collected test data. In order to limit the size of the archive, we removed the images provided by the contest and generated for training.

So please copy all contest images into data/ISS, data/Lab, data/Lab2, data/Lab3, data/Sim folders.

If you wish to generate training image for the primary object classifier and secondary object classifier, take a look at `Utility::exportObjectsInformationsToImages` and `Utility::exportPrimaryObjectsToImages`.

B. Octave scripts

1. Classifiers training scripts

Classifiers training scripts are always organised in the same way. Take a look at octave/secondary_object_classifier:

- 1-COLLECT folder allows you to transform images to octave code (bundle).
- 2-PCA folder allows you to apply PCA on the bundle (that you must copy from the collect folder). You need to look first at the `process.m` file. The `generateC.m` file allows you

to transform eigenvectors into a c++ class.

- 3-TRAINING folder allows you to train neural networks. You need to look first at `training.m`. The `generateC.m` file allows you to transform neural network weight into a c++ class.

2. Shift linear regression scripts

The linear regressions described on III. F. 2. is calculated in the octave/decals scripts. The `collectData.m` file allows us to import datafiles generated by our main program into octave data. The `process.m` file allows us to calculate the linear regression.

C. Bundle generation script

The `generate_bundle.php` reads the `bundle_config.php` file in order to generate a bundle (which we can copy and submit to TopCoder) from specified files. To generate a bundle, you need to install php and then simply execute `php generate_bundle.php`.

V. Conclusion

Thanks for this exciting contest :).