Université Pierre et Marie Curie

Master IMA
PIMA

Project report

Student : Mario Viti
mario.viti@etu.upmc.fr
Teacher : Nicolas Bredeche

28 mai 2017

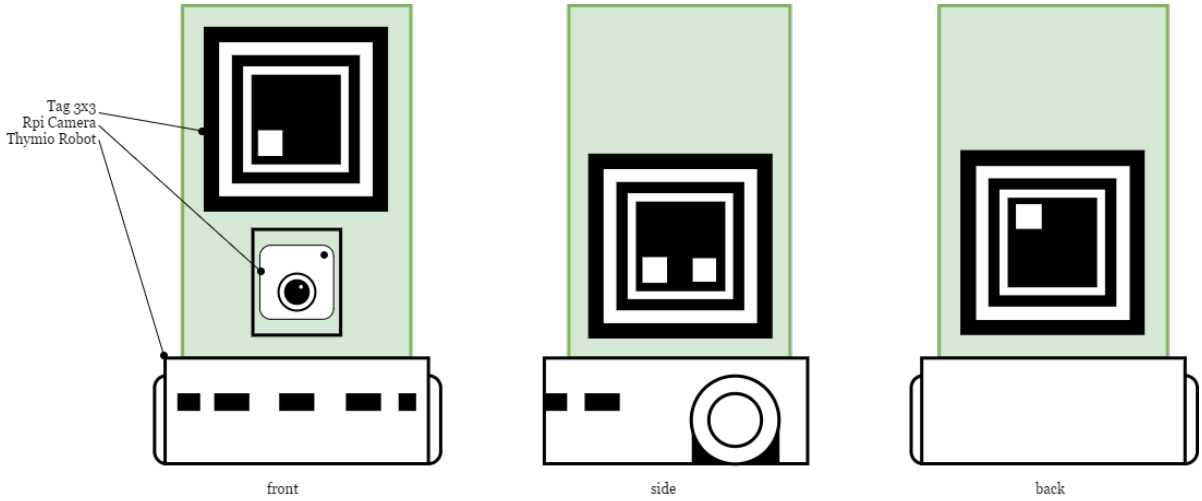

FIGURE 1 – Thymio robot with Raspberry, Camera and Tag.

0.1 Project presentation

0.1.1 Presentation

Realization of an embedded solution for Raspberry [1] powered Thymio robots. Thymio-2 were originally designed at EPFL [2], and mostly targeted for educational use. They're cheap and reliable, and have been extended with an inexpensive but powerful Raspberry PI 3, plus an on-board camera and battery. These extended Thymios are primarily used social learning experiments for which it has been developed an ad-hoc framework.

0.1.2 ThymioPYPI framework

ThymioPYPI [3] is a framework for programming a Thymio2 with a Raspberry PI in Python 2.X. Experiments are the programs launched on the Raspberries by the framework and constitute the specific behaviours desired for the robots. As such, a minimal implementation of an experiment corresponds to coding the behaviour of the robot at each step of the experiment. Thymio PYPI provides a network infrastructure shipping software to the Thymios and monitor the distributed system.

0.1.3 Goal

To test these hypothesis an implementation of a rudimental view system must be put in place in order for Thymios to extract information from the sorrounding environment and experience collective behavoiours. The main product of the PIMA project is a support module program for all experiment that involves visions tasks, like recognition of obstacles and other Thymios. This slave application must run transparently and at a minimum rate of $10hz$. This constraint must be assured in order to avoid making action inconsistent with the sorrounding environment. In order to achieve this goal, implemntation choices need to take into account the threadoff between performance and precision.

1. https ://www.raspberrypi.org/documentation/
2. https ://www.thymio.org/fr :thymio
3. https ://github.com/nekonaute/thymioPYPI

0.2 Tag Detection in real time.

In this session it will be proposed a formalization of a realtime image processing application or detection task.

A realtime image processing detection task preformed on a stream of frames coming from a sensor of any kind can be defined as a service or slave or server application. The application complies to provide correct output within the other sole constraint of maximum time of service *tos* or minimum frequency of service *fos* imposed by the master or client. Therefore it can be defined by the set {outputs,tos}.

In *Tag Detction* the correct outputs are : the contours of the tags present in the images, the identificators of each tag and the estimated distance from the camera of each tag.

In orfer to minimize *tos* and maximize *fos* a realtime processing algorithm can be divided in 2 phases : a *fine analyse* and a *coarse analyse*.

0.2.0.1 Fine analyse.

A fine analyse consists of the best effort analyse of the current frame, this step is usually more computationally intensive than the coarse analyse. In our case **Algorithm 1** is the fine analyse.

0.2.0.2 Coarse analyse.

The coarse analyse reuses data acquired from the fine analyse, because as an hypothesis we assume that ideally little amount of change occours from frame to frame, therefore previous informations are still valid. In our case **Algorithm 2** is the coarse analyse.

Algorithm 3 combines the two phases analyse and merges the results.

Data: currentFrame
Result: tagIDs, tagCountours, tagDistances
edgeImage = Canny(currentFrame);
tagCountours = chekAreaRatio(edgeImage);
tagImages = rectifyImage(currentFrame,tagCoutours);
tagIDs = readID(tagImages);
tagDistances = estimateDistances(tagCoutours);

Algorithm 1: tagDetection

Data: previousTagContours,previousFrame,currentFrame
Result: tagCountours, tagDistances
tagContours = LKpyramid(previousFrame,currentFrame,previousTagContours);
tagDistances = estimateDistances(tagCoutours);

Algorithm 2: motionDetecion

```
frameCounter = 0;
frameBufferingLenght = 2;
Global running;
while running do
    currentFrame = readFrameFromCamera();
    if frameCounter%frameBufferLenght == 0 then
        tagIDs, tagCountours, tagDistances = tagDetection(currentFrame);
    else
        tagCountours, tagDistances = motionDetecion(tagCountours,currentFrame,previousFrame);
    end
    frameCounter = (frameCounter+1)%frameBufferLenght;
    previousFrame = CurrentFrame;
end
```

Algorithm 3: postProcessing

The resulting *tos* will be $tos_{avg} = \frac{tos_{fine}+tos_{coarse}}{2}$ with $tos_{coarse} < tos_{fine}$ we'll achieve better performances on average.

0.3 Software Development

For the implementation of the algorithms it has been developed a *Python* package to be integrated in the *ThymioPYPI* framework called *camera_ tools*.

The software is designed as a **metaDetector**.

The metaDetector design abstraction is defined at multiple levels : it is a set of functions { *preProcessing*,*postProcessing* } and the output of the *preProcessing* is piped to the *postProcessing* function as shown in **Algorithm 4**. Furthermore it is an **Interface** for image processing implemented as **Class**, tag detection is subsequently realized by *implementing* a *metaDetector*. The nature of this division, the details of the preProcessing and postProcessing will be discussed further in the next session.

Data: frame
Result: postProcessingOutput
preProcessingOutput = preProcessing(frame);
postProcessingOutput = postProcessing(preProcessingOutput);

Algorithm 4: metaDetector

The most important part of the package are :

camera_tools package :

- CameraController : handles RaspberryPi camera module PiCamera.
- ImageProcessor : holds the CameraController and handles the buffer in which pixels are stored by applying preProcessing and postProcessing to it.
  metaDetector : exposes the ImageProcessor capabilities by defining the metaDetector Interface.
- TagDetector : implements the metaDetector interface.
- image_utils : collection of functions for image processing.
- tag_recognition : collection of functions for tag detection.

For a more detailed description on the usage of the software refer to the *thymioPYPI manual*[4].

4. https ://github.com/nekonaute/thymioPYPI/blob/master/Robocologie$_m$*anual.odt*

0.3.0.1 Realtime image processing.

The *Raspberry 3 b* is equipped with a *1.2 GHz 64-bit quad-core ARM Cortex-A53* which allows different threads to execute on different cores. A realtime application can exploit multithreading by definig a set of independet tasks. We'll define 3 types of independent tasks in our application.

1. **capture** : this task is mostly camera dependent and it evolves capturing the image from the camera and converting to readable format.
2. **preProcessing** : in the previous section this was described as one part of the meta algorithm for image processing, it refers to any task that involves "non heavy" (linear) computation on images i.e. RGB to HSV conversion .
3. **postProcessing** : in the previous section this was described as one part of the meta algorithm for image processing, it refers to any task that involves "heavy" (non linear) computation on images i.e. EdgeDetection involving Convolution.

This set of tasks defines a general realtime application running on a on board solution with or wihtout GPU support to maximize throughput.

Figure 2, shows how the 3 tasks need to be secured by locking critical session on shared buffer among threads. *PreProcessing* function computional requirement is a suggestion as it's applyed to a shared buffer, sharing buffers limits memory usage but serializes computation. As an hypothesis keeping linear computation to the *PreProcessing* step will ensure a capture request returning after *PreProcessing* ended and a copy of the buffer is kept for *postProcessing*.

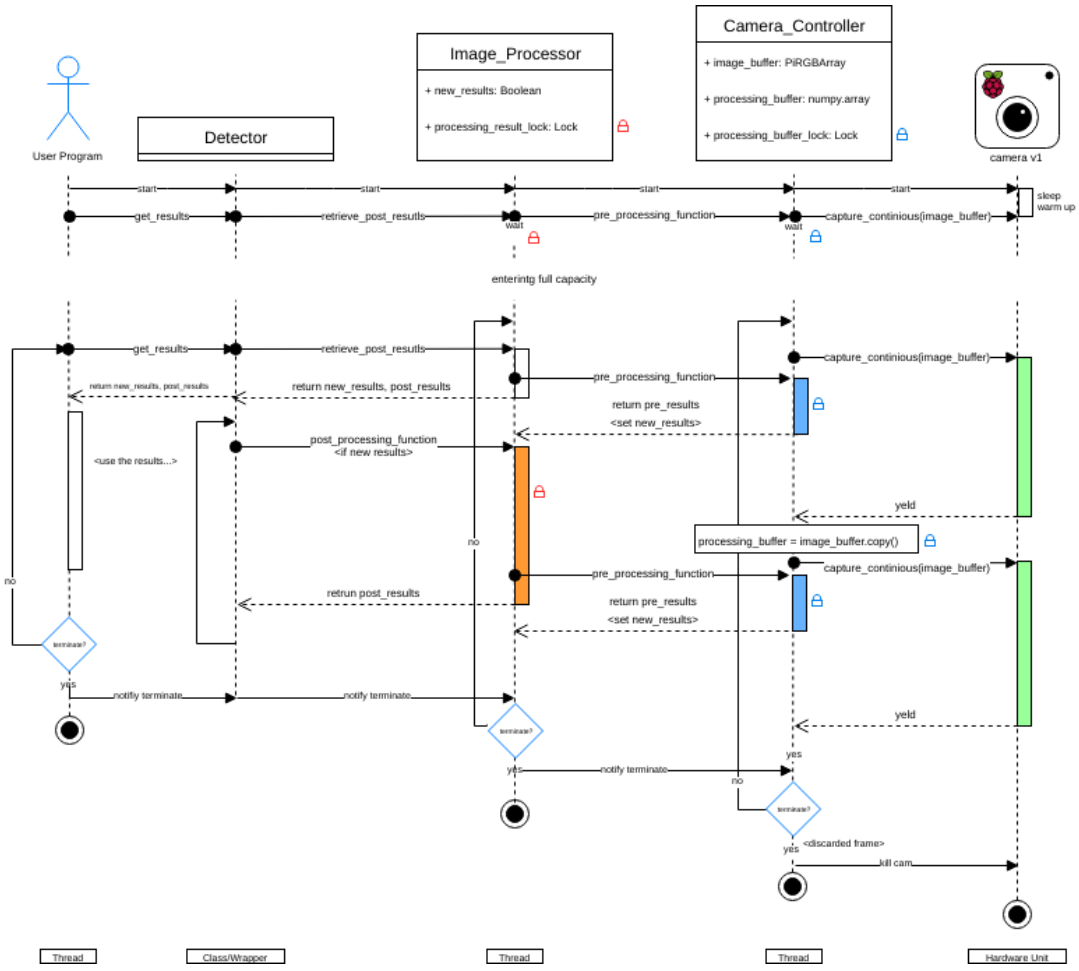

FIGURE 2 – lifecycle of a image processing realtime application.

0.4 Tags

0.4.1 Encoding

An encoded tag is modeled as a discreete bi-dimensional serie $e(x,y) = \sum_{i,j}^{n,m} a_{i,j} h(x - iS_b, y - jS_b), n, m \in \mathbb{N}$ with $h(x,y) = Arect_{S_b}(x - S_b/2, y - S_b/2)$ and T_b the *space per bit* and $a_{i,j} \in \{-1, 1\}$, the wave shape $h(x,y)$ and the simbol $a_{i,j}$ will encode the information in binary format. For better precision $m - ary$ encoding are better avoided because of the high noise in the communication channel.

The noise in the communication channel is modeled as an additive white gaussian noise or **AWGN** with constant dsp equal to N_0. In a one dimensional encoding while in presence of this noise applied to a signal just described the signal to noise ratio **SNR** of an SLI filter or receptor $g_r(t)$ is maximixed by applying the *Cauchy-Swartz* theorem.

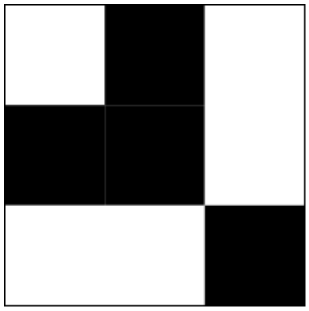

FIGURE 3 – an example of a 3x3 bits encoded tag.

$$SNR = \frac{\int_{-\infty}^{+\infty} g_r(t) h(t_0 - t) dt}{\sqrt{N_0 \int_{-\infty}^{+\infty} g_r(t)^2 dt}} \leq \frac{\sqrt{\int_{-\infty}^{+\infty} h(t_0 - t)^2 dt}}{\sqrt{N_0}}$$

From which it is derived the analitical form of the optimal adaptive filter $g_r(t) \propto h(t_0 - t)$, this can be applied to the 2D case without loss of generality $gr(x,y) \propto h(x,y)$.

0.4.2 Reading

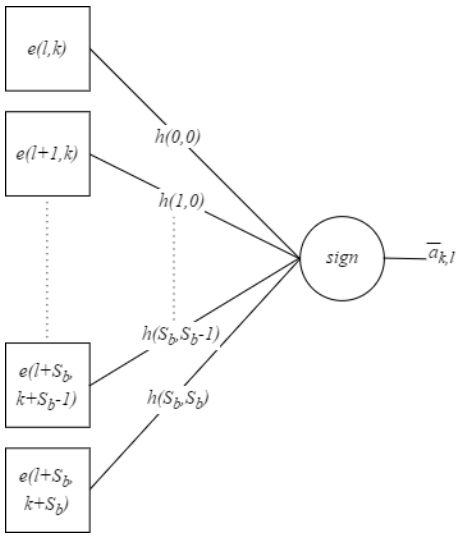

FIGURE 4 – perceptron reading one bit.

The sequence of symbols received $\hat{a}_{k,l}$ is the result of the sampling of the signal processed by the adaptive filter which consists of a utile term and a noise term. By means of the adaptive filter the utile term will be greater than the noise term, ideally.

$$\hat{a}_{l,k} = sign((h * h)(S_b/2, S_b/2) a_{l,k} + (n * h)(l,k))$$

with $n(x,y)$ being the **AWGN** random signal and $(h * h)(T_b/2, T_b/2) a_{l,k} > (n * h)(l,k)$ respectively the utile and noise term.

The discreete version of $(h * h)(T_b/2, T_b/2) a_{l,k}$ is a dot product $\sum_{i,j}^{S_b,S_b} Arect_{S_b}(i - S_b/2, j - S_b/2) e(l+i, k+j)$.

The computation involved in this "decisional process" is exactly the same that occours when using a perceptron. In this case however the *weitghts* for the linear classification ar not learned but computed anallitically, as in theory this is the optimal filter, a perceptron will tend to this solution.

0.5 Tracking Evaluation

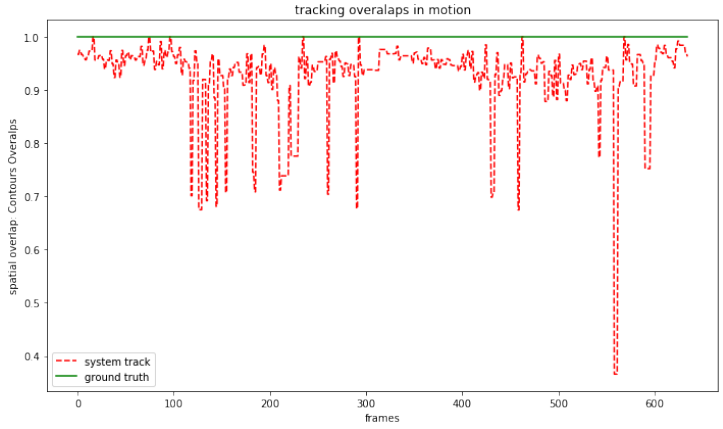

FIGURE 5 – different experimental settings.

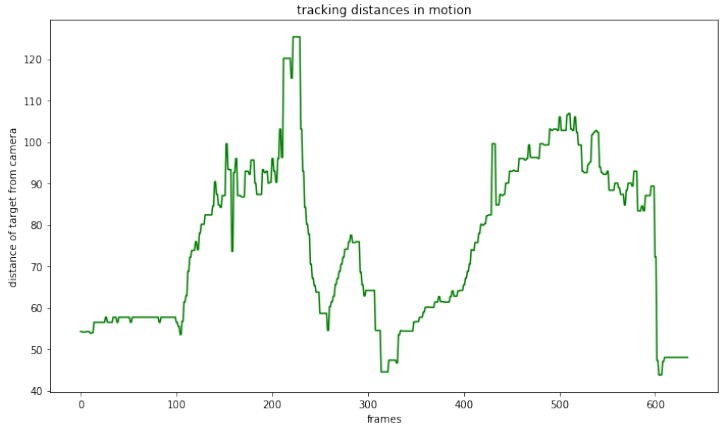

FIGURE 6 – different experimental settings.

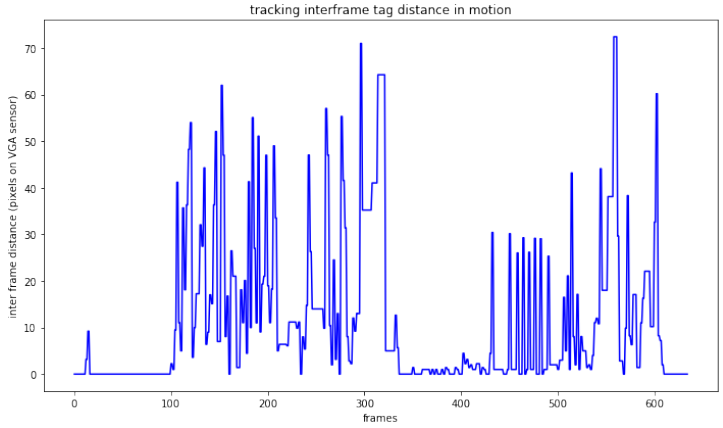

FIGURE 7 – different experimental settings.

This tag detection application can be evaluated as a **tracking system** as desrcibed in[4]. We will use [4] as a guideline for metrics and performance estimation practices as these has been used on other popular tracking systems. As described in [4] *"First we will define the concepts of spatial and temporal overlap between tracks, which are required to quantify the level of matching between Ground Truth (GT) tracks and System (ST) tracks, both in space and time. The spatial overlap is defined as the overlapping level $A(GTi, STi)$ between GTi and STi tracks in a specific frame k :"*

$$A(GTik, STik) = \frac{Area(GTik \cap STik}{Area(GTik \cup STik}$$

In order to evaulate the performances an experiment has been set up in the arena with Thymio robots. A robot moved in front of the camera randomly, the tracking is executed once by using the *fine analyse* on each frame (GT), subsequently the tracking is done by intervealing *fine analyse* and *coarse analyse*(ST) based on motion estimation, with the cotours overlaps are computed for each frame (Figure 5). For each frame, distance from the camera (Figure 6) and interframe distance (Figure 7) are registered in order to correlate how distance from the camera and velocity affects the tracking system.

False alarm track (FAT) of False Positive (FP) :

[4] *"Although it is easy for human operators to realise what is a false alarm track (event) even in complex situation, it is hard for an automated system to do so. Here, we give a practical definition of false alarm track".*

$$FAT = \frac{\sum_{k=0}^{n} A(GTik, STik)}{n} < thr$$

, n = number of frames, $thr = 0.6$.

Track Matching Error (TME) :

[4] *"This metric measures the positional error of system tracks. TME is the average distance error between a system track and the GT track. The smaller the TME, the better the accuracy of the system track will be."* In this application the distance is normalized with the maximal distance possible as the diagonal of the VGA window :

$$TME = \frac{\sum_{k=0}^{n} Dist(GTik, STjk)}{n(VGA_d)}$$

[4] *"where Dist() is the Euclidean distance between the centroids of GT and the system track : TMED is the standard deviation of distance errors"* In Figure 8 two tracks are overalapped as they appear on screen, specifically only the center of the contour is registered.

— FAT : 0.005
— TME : 0.02
— TMED : 0.01

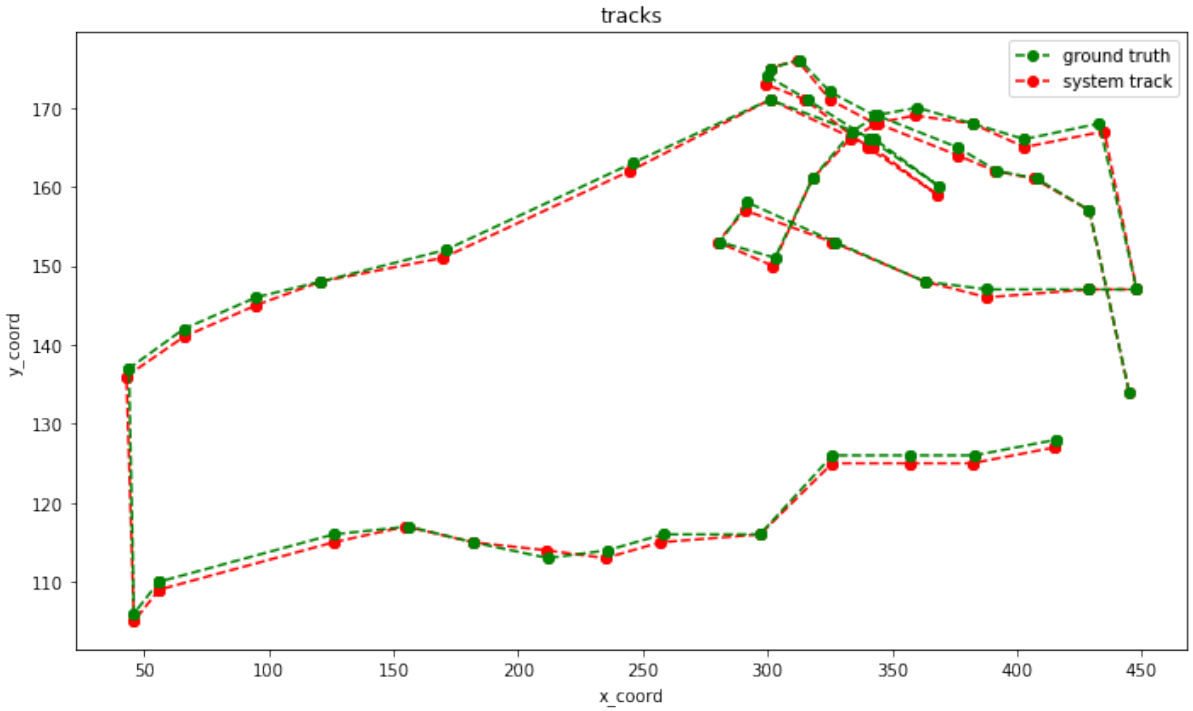

FIGURE 8 – different experimental settings.

0.6 Experimental Validation

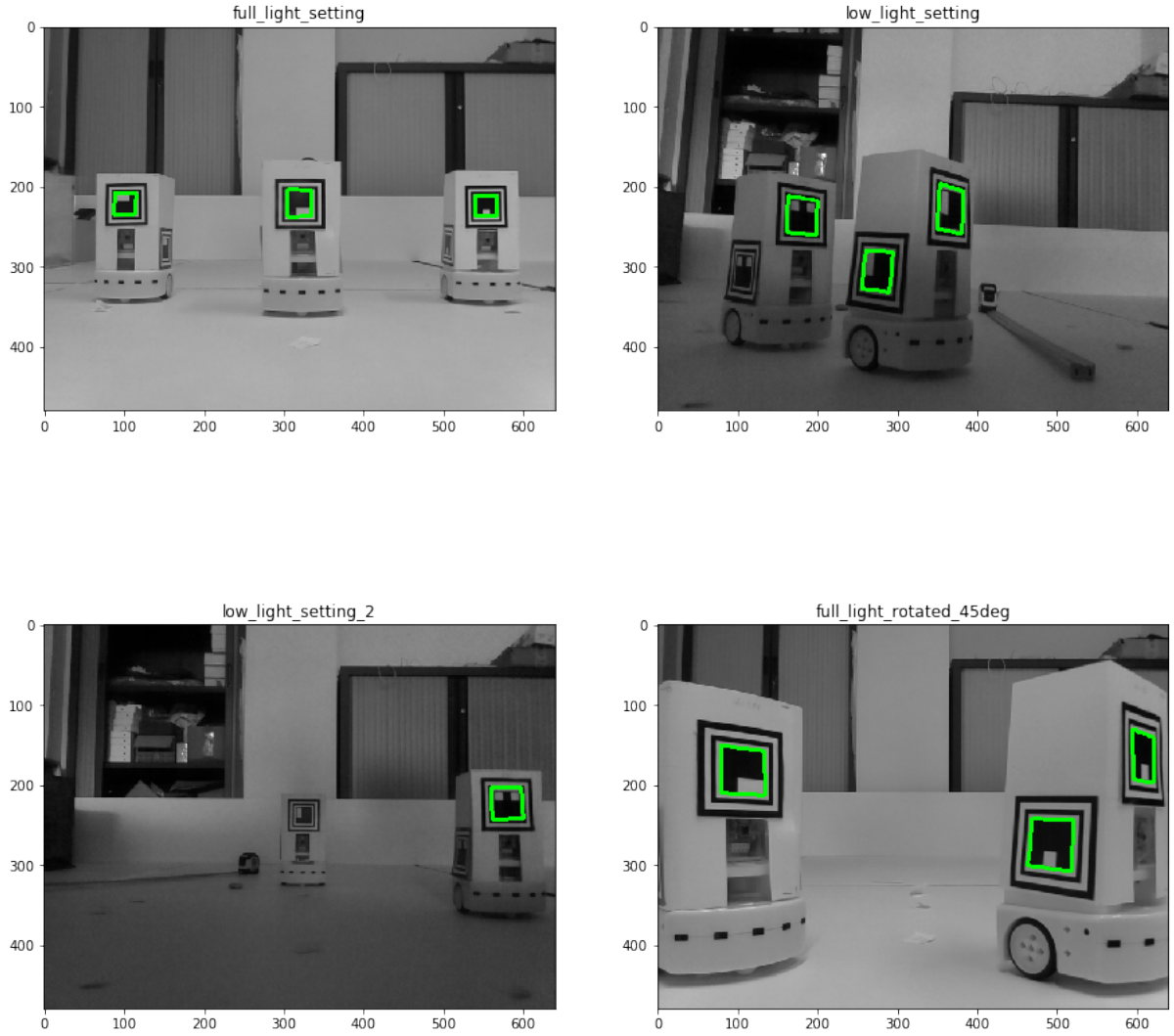

FIGURE 9 – different experimental settings.

In order to evaulate the performances an experiment has been set up in the arena with Thymio robots. Earch robot was positioned in a precise location and it was shown to the camera. This experiment is repeated in two different settings, with a full light setting with homogeneus light coming from artificial lamps and a low light settings wiht a noisy background, non homogeneus light and a gradient like soft shadow overlaying the scene. It is one last time repeated by changing the rotation with respecto ro the camera of 45°.

In order to measure the performance it has been defined a per second *frequency of detecion* as $\frac{hit_t}{max_t(hit_t)}$ where a *hit* occours when tag t is detected a number hit_t of times per second and $max_t(hit_t) = fos$.

Caveat. This results are contingent to the setting : the size of the tag and the definition of the sensor influence the results as performance are in funtion of the area in pixels of the tag on the sensor which is a function of the tag real dimensions and the camera definition.

Settings :
- tag innermost side size : 4cm
- sensor definition : VGA (640,480)

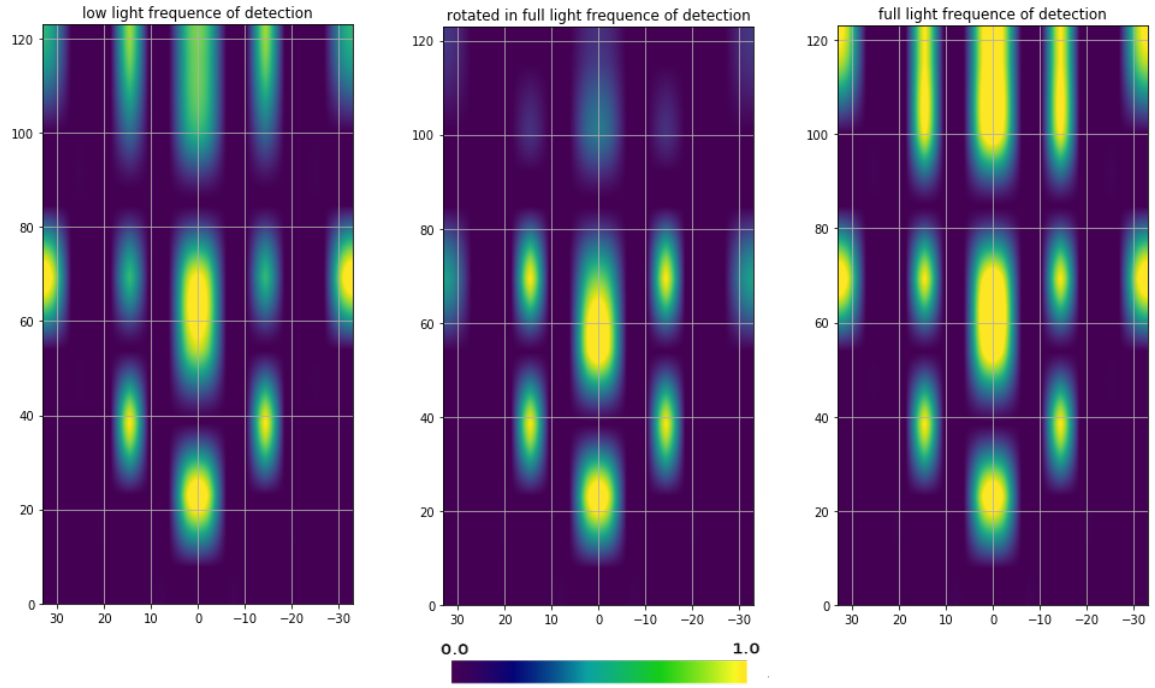

FIGURE 10 – frequencies of detecion against positions (in cm).

Distance estimation has been evaluated agaist real distance from the camera. In presence of low light and noise it's noticeable an higher variance in the error. Also it seems that a systematic bias is present in distance estimation when distance augments this is probably due to the sampling of a continuos shape on a discrete grid which makes the apparent size different from the real size.

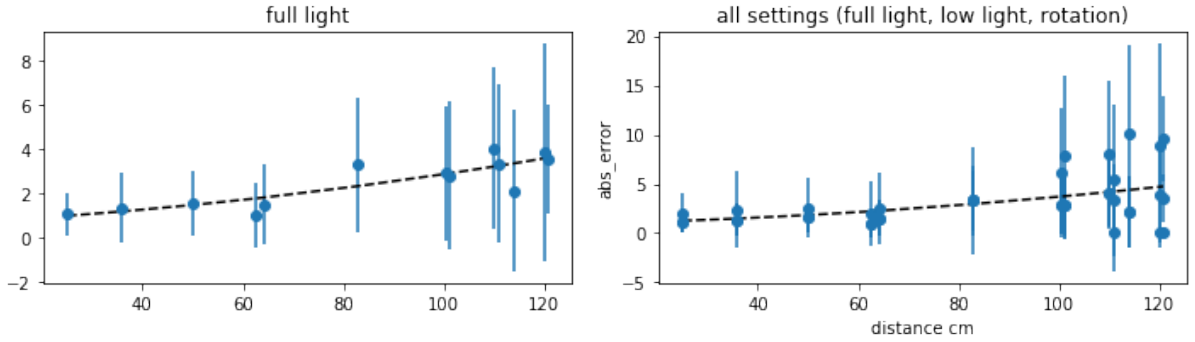

FIGURE 11 – absolute error of real and estimated distance againts real distance, bars indicates variance in the absolute error.

Bibliographie

[1] J. Canny, "A Computational Approach to Edge Detection," vol. PAMI-8, no. 6, pp. 679-698, Nov. 1986.

[2] JJ.R. Bergen, P. Anandan, K. J. Hanna, R. Hingorani, "Hierarchical model-based motion estimation", Computer Vision - ECCV'92 pp. 237-252, May 1992.

[3] M. Hara, M. Watabe, T. Nojiri, T. Nagaya, Y. Uchiyama, "Optically readable twodimensional code and method and apparatus using the same" US 5 726 435, 10 mar 1998.

[4] F. Yin, D. Makris, S. Velastin and Digital Imaging Research Centre, Kingston University London, UK "Performance Evaluation of Object Tracking Algorithms" Computer Vision – ACCV 2006, pp. 151–161, Jan. 2006.

[5] N. Bredeche, E. Haasdijk, A. E. Eiben, On-Line, "On-Board Evolution of Robot Controllers", in Artifical Evolution : 9th International Conference, Evolution Artificielle, EA, 2009, Strasbourg, France, pp. 110-121, Oct 2009.

[6] J.M. Montanier, S. Carrignon, N. Bredeche "Behavioral specialization in embodied evolutionary robotics : Why so Difficult ?" Front. Robot. AI vol.3 art 38, Jul 2016.

[7] OpenCV Computer Vision Library. [Online]. Accessible : http ://opencv.org/ [Accessed :23-May-2017].