# Object Recognition and People Tracking in a home environment

Ismail Lahkim Bennani

August 22, 2017

## Contents



## Introduction

## I. Challenge 1: Storing groceries

This test focuses on the detection and recognition of objects and their features, as well as object manipulation. This is the scenario as described by the rulebook:

Setup:

1. **Location:** One of the bookcases or cupboards in the apartment is used for this test, one where a table is near or can be put.

2. **Start position:** The robot will start between the cupboard and the table in a random orientation, but facing towards the Cupboard. .

3. **Cupboard:** The cupboard has 5 shelves between 0.30m and 1.80m from the ground and contains several objects.

   – **Door:** The cupboard has a single door, which is closed initially. This door encloses some of the objects, covering up to one half of the cupboard (e.g. the left or bottom half), as indicated by the hatched area in the image.

4. **Table:** A table near to the Cupboard has 10 objects. If not all fit the on the table, they will be added during the test. The maximum distance between the Table and the Cupboard is 2m.

5. **Objects:** Objects on the Cupboard and on the Table can be known, alike, or unknown. Also, there will be more than one object in each shelf.

Task:

1. **Opening door:** The robot starts opening the Cupboard's door. If the robot is unable to open the door, it may ask the Referee to do it instead.

2. **Cupboard inspection:** The robot inspects the cupboard locating and categorizing existing groceries.

3. **Finding the table:** The turns around and locates the table.

4. **Table inspection:** The robot approaches the table starts analyzing the newly bought groceries (i.e. objects).

5. **Moving objects:** The robot chooses which object to move first from the Table to the Cupboard, allocating similar objects all together.

   - Objects of the same type (i.e. identical known objects or akin alike objects) must be placed one next to the other.
   - If the Cupboard has no object of the same type, then objects must be grouped by category (e.g. drinks with drinks, snacks with snacks, etc)
   - If the Cupboard has no similar object, the robot must clearly state its decision on how to solve the problem. For instance, the robot can define a place for the newly found Category (e.g. Food was found but there is no other food in the cupboard), or group all new objects together (e.g. placing all Unknown objects together).

   **Note:** Either before or after grasping an object the robot may announce the name of the object found.

6. **Repeat:** This repeats until the time is up or all groceries are stored.

For this challenge I worked on the object recognition programs.

## I.1. Object recognition

The core component of the perception module is the one used for object recognition. We do that by using two complementary approaches: a deep convolutional network called YOLO[3] and a clustering of the objects based on our depth image.

YOLO is powerful at discriminating between a wide range of objects given a proper dataset while the depth-based approach doesn't need any training.

The combination of these two methods give use a robust way of finding known and unknown objects in an RGB-D image. We can then use the 2D information we get from the recognition to find the location and size of the objects in the real world.

### I.1.a. YOLO

YOLO (and more specifically YOLOv2) is considered as one of the most performing neural network for object recognition. It was used by 12 out of 15 teams in RoboCup@Home2017 and by more than half of the teams in all the RoboCup competitions. It takes an image as input and outputs labeled bounding boxes corresponding to the recognized objects (see Fig. 1)

We decided to implement our perception module in python but YOLO has been implemented by it's author in C. I found several other implementations in python using either caffe or tensorflow but none of them was as fast as the original version. The C version was 5 to 10 times faster than the others, so

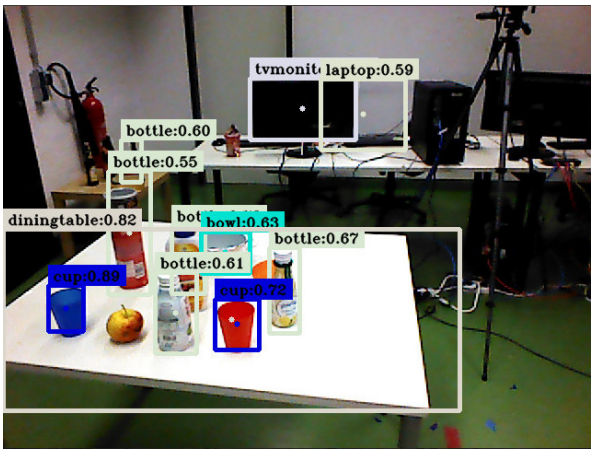



**Figure 1:** YOLO predictions

I decided to create a kind of bridge between that and the rest of the code. Concretely, the python class I created spawns a new process to run YOLO and communicates with it through pipes.

I used a Nvidia GTX 1060 for my tests, the original YOLO can process an image in 100 to 150 milliseconds while the python class I use does it in 110 to 180 milliseconds.

However, this approach has two major problems. One of our biggest constraints was processing power: our robot had no graphics card so we couldn't use it to run a deep neural network. We had to use an external computer mounted on the robot to do our computations (see Fig. ...). At first we wanted to build a desktop-like computer for this purpose but we realised that it would be too difficult to cool it in such a small space, and most importantly that it would be nearly impossible to power it. We were not allowed by the company that lent us the robot to use its batteries to power anything else than the robots and we didn't have enough room to put a battery big enough to power a computer powerful enough for our purposes without using custom components. We decided to use a laptop computer and the most powerful we had was using an Nvidia GTX 1060.

The second major problem is that YOLO is not able to recognize objects that were not part of the training dataset.

### I.1.b. Depth clusters

To solve the unknown object problem, we decided to use the depth point cloud that we got from our depth camera to recognize every potential object in our field of view.

To do so we start by keeping the points in front of us and by removing all the big planar components from our point cloud, that corresponds to the floor, the walls, the tables etc.. Then we cluster the remaining points by Euclidian distance. Two points are part of the same cluster if they are close enough to each other (see Fig. 2). Finally we compute the 2D projection of the cluster position on the RGB image to get a bounding box of the unknown object (see Fig. 3).

In practice, we could only use this method to find the objects on a table when we were looking at the table. When we were looking at the shelf for example, we had too many wrong predictions:

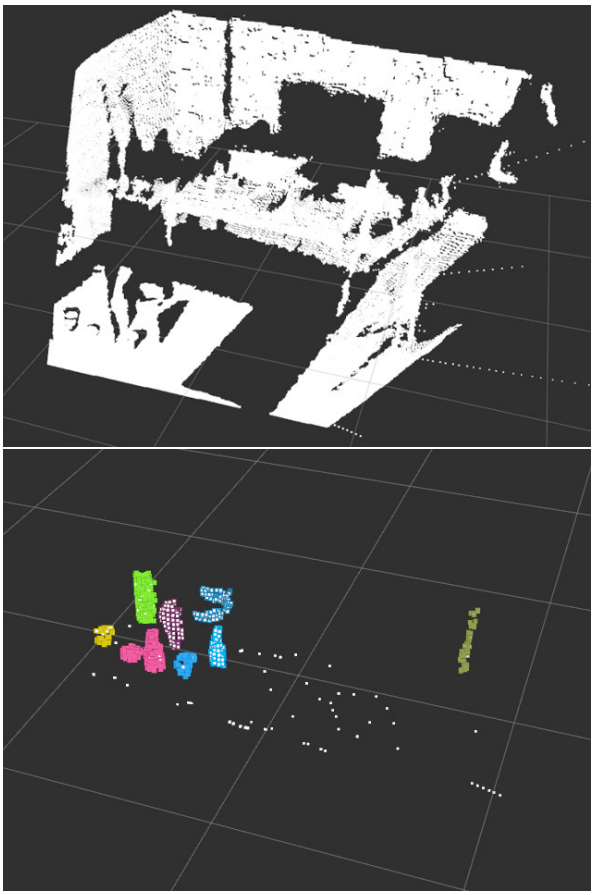


**Figure 2:** Point cloud (top) and object clusters on the table (bottom)


**Figure 3:** 3D clusters projected to the 2D image as unknown objects

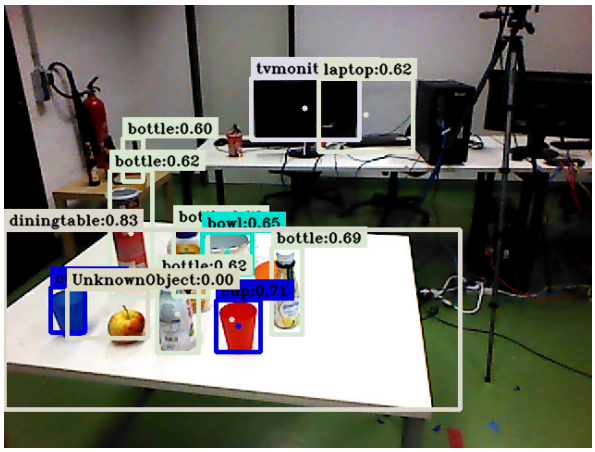



**Figure 4:** YOLO predictions (Fig.1) and depth clusters (Fig.3) merged

the rims of the shelf were unknown objects, some pieces of the wall were not big enough on our point cloud (because of partial occlusion of the wall by other objects) to be removed as big planes, so they were objects too. However, that was not a big issue since we only care about the unknown objects for grasping them, and in our scenario those objects are on the table.

With this method we find most of the objects on the scene, even those that YOLO recognizes. We need to merge the two approaches results together, to do so we compare each couple of predictions (one from YOLO, the other from the depth image) and we compute the intersection over union of the area of their bounding box. If the result is close enough to 1, it's the same detection (see Fig. 4).

This method was first intended to find the unknown objects in the challenge scenario, but we realised that we could also use it to enhance our recognition. I added an extra step to this approach: classification. Once we get the 2D bounding box in the RGB image, we compute a hue histogram of the object's image and we run it through an SVM to identify it (see Fig. ...).

## I.2. From 2D to 3D

Once we have our 2D bounding boxes, we need to find the position of the objects in the world to be able to interact with them. We use the depth image for that (see Fig. 5).

The depth image is a float32 matrice where each cell is the depth of the corresponding pixel with respect to the camera (located in the robot's head). Given a 2D bounding box, we crop the image, we compute a histogram of the depth values and we down sample it to 5cm wide bins, then we select the bin that contains the most points. This method is a lot more robust than taking a the depth of a single pixel in the bounding box (our first solution was to take the depth value of the pixel in the center of the bounding box). We can see in Fig. 5 that the leftmost cup is almost black in the depth image, a black pixel means that we don't have any depth value for that pixel and that happens a lot with transparent or semi-transparent objects.

We can also estimate the apparent size of an object with the informations we have. We compute the size ratio of the bounding box to the size of the image, then with the horizontal and vertical field of view of the camera we can compute the angular size of the object in the image with respect to the camera and we can find the actual size of the object thanks to it's depth relative to the camera. With this method we find the apparent width and height of the object, we cannot find its third dimension but we don't need it.

The size we compute is a rough approximation (I'm approximately 1.77m tall and the recognition gives me a height of approximately 1.50m) but we don't really need a high precision: the size is used to find what object is graspable based on the maximal width of our robot's gripper. For the actual grasp, the robot uses an artificial skin (developed by ICS,

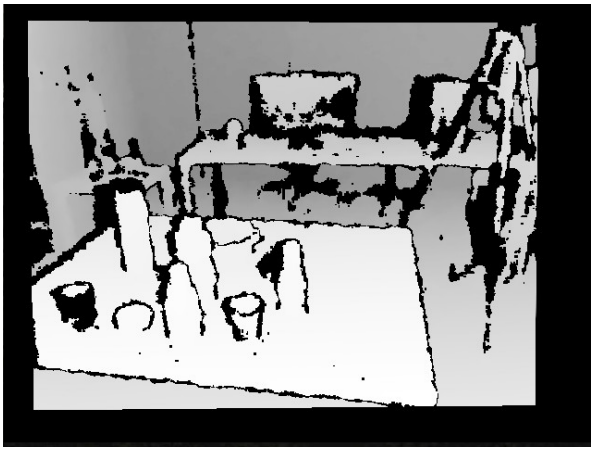


**Figure 5:** Depth image

it has a proximity sensor amongst other sensors) to adjust the gripper's position and size to the object's actual size.

The final result of the recognition module (see appendix .1.) is the label, position and size of each recognized object relative to a fixed coordinate frame (see Fig. 6).

### I.3. Results

Our method for solving the challenge is the following. Note that we had some time before the challenge to map the room and to set some predefined positions.

1. Go in front of the shelf
2. Ask the operator to open the door
3. Recognize the objects in the shelf for some frames and keep the stable recognitions (sometimes we have noise in the recognition)
4. Cluster the objects in the shelf based on their semantic class and their position
5. Compute available empty space next to each cluster
6. Go to the table
7. Recognize the objects in the table for some frames and keep the stable recognitions

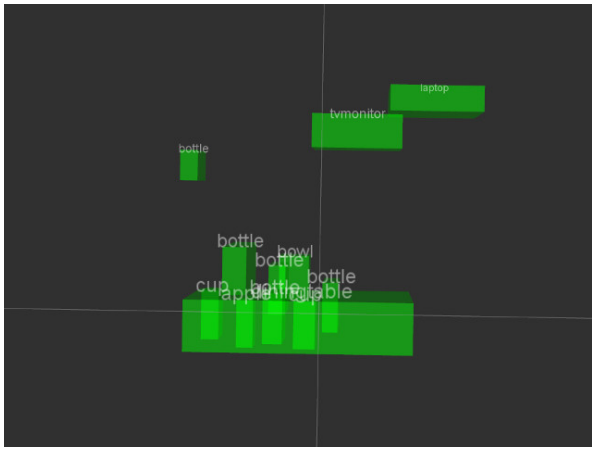



**Figure 6:** Recognition result

8. Choose an object to grasp and grasp it: the choice is based on the object itself (a bottle is easier to grasp than a bowl), the object position (start with the closest ones) and the clusters found in the shelf (don't grasp an object if you don't know where to put it)
9. Go back to the shelf
10. Compute a proper placing position: the choice is based on the empty space computed before.
11. Place the object
12. Repeat from step 6 until there is no more object on the table

The main problem we encountered during the competition was the time it took us to perform the tasks. We had 1:30 minutes to grasp the first object on the table, and it took us 1:35 minutes to do it, mainly because we lost too much time recognizing every object on the shelf (the shelf is too big to see it in one picture, we had to move our head several times, see Fig. 7) and computing clusters and proper placing positions.

This problem was common to all the team, actually there is only one team that managed to grasp the object in the given time but they couldn't place it in the shelf, we managed to get the third place (out of 15) thanks to the objects we recognized on the shelf. The day after the challenge, we worked on reducing the delays (the 1:30 minute rules was not part of the original rulebook, it was added 1 hour before the actual challenge) and we managed to grasp and place 1 object in the given time.

## II. Challenge 2: Help me carry

This test focuses on safe, robust navigation, people following and navigation in unknown environments. This is the scenario as described by the rulebook:

Setup:

1. **Location:** One of the arenas (apartment) and its surroundings. The apartment is in its normal state. Part of the test is performed outside the arena in a public space.

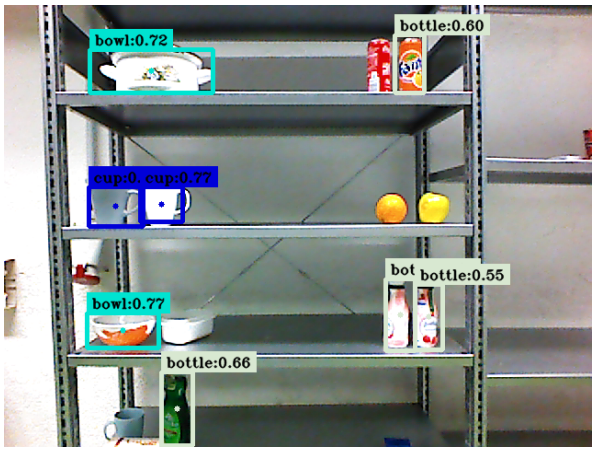



**Figure 7:** Shelf objects recognition

2. **Start:** The robot starts waiting inside the arena.

3. **Car:** The car is any landmark chosen (but *not* announced) beforehand outside the arena. Several bags with groceries are located where the car is parked.

4. **Doors:** All doors in the apartment are initially open.

5. **Operator:** A professional operator is selected by the TC to act as the operator of the robot.

6. **Uncontrolled environment:** There are no restrictions on other people walking by or standing around throughout the complete task.

Task:

1. **Start:** The robot starts at a designated starting position in the arena, and waits for the *professional* operator. The operator steps in front of the robot and tells it to follow (e.g. by saying follow me). The team is *not* allowed to instruct the operator.

2. **Memorizing the operator:** The robot has to memorize the operator. During this phase, the robot may instruct the operator to follow a certain setup procedure.

3. **Following the operator:** When the robot signals that it is ready to start, the operator starts walking –in a natural way– towards the car. Upon arrival, the operator will indicate the robot when they have reached their destination as instructed by the robot (e.g. by saying here is the car or stop following me).

4. **Bring the groceries in**
   The robot is asked to deliver a bag with groceries to a specific location (e.g. Take this bag to the kitchen table).

   (a) **Bag pick-up:** The robots gets the bag. For this there are several options to achieve this: **a)** Human puts bag in robot's hand, **b)** robot picks up bag on floor, **c)** Robot takes bag from operator's hand

   (b) **Bag delivery:** The robot delivers the bag to the instructed destination. It may place the bag on the floor or onto the placement location.

   (c) **Asking for help:** Close to the delivery location is another person. The robot must face at them and kindly ask them to help carrying groceries into the house.

5. **Find a person:** After reaching the designated room, the robot needs to find a person (there is only one person in the room, the name is meaningless).

**Memorizing the *new* operator:** The robot has to memorize the operator that will help. During this phase, the robot may instruct the operator to follow a certain setup procedure.

**Guiding the operator:** When the robot signals that it is ready to start guiding, the robot guiding the operator to the car. The robot must clearly announce when the destination (the car) is reached.

[leftmargin=3cm]**Closed door:** Along it's path to the car, the robot will find a closed door (most likely the entrance to the house) that will need to be opened to reach the destination.

For this challenge I used the previous work to recognize people and I worked on the tracking programs.

## II.1. Tracker module

Since our Object Recognition module is able to detect people as any other object, we use it as our people detector for the tracker. The deep neural network that we use is very good at finding people, it can even recognize the reflection of a person on a semi-reflective surface such as a monitor !

However, we want to do more than detection, we want to track. For that we need to match all the detections of people that we have between the frames. We do that by considering our problem as a stable marriage problem. In addition to that, we need to identify the people that we see as "operator" or "not operator" to solve the occlusion problem for example, or to detect that we don't see our operator anymore. Our approach to solving this problem is to use feature descriptors such as hue histograms and SVMs.

### II.1.a. Stable marriage problem

**Definition** The stable marriage problem is the problem of finding a stable matching between two equally sized sets of elements given an ordering of preferences for each element.

A *matching* between two sets $A$ and $B$ is a mapping from the elements from $A$ to the elements of $B$. A matching is *not* stable if:

- There is an element $a$ of the first matched set which prefers some given element $b$ of the second matched set over the element to which A is already matched, and
- $b$ also prefers A over the element to which $b$ is already matched.

In other words, a matching is stable if we cannot find two couples in $A \times B$ in which two of the elements would rather be together than with their respective partners.

**Implementation** In our case, we want to match the people detections from frame $n-1$ to the people detections from frame $n$. Our elements are the detections and we use the 3D euclidian distance between the position of the people we detect (that we get thanks to the recognition module) to order the elements. The closer they are to each other, the more likely they correspond to the same person.

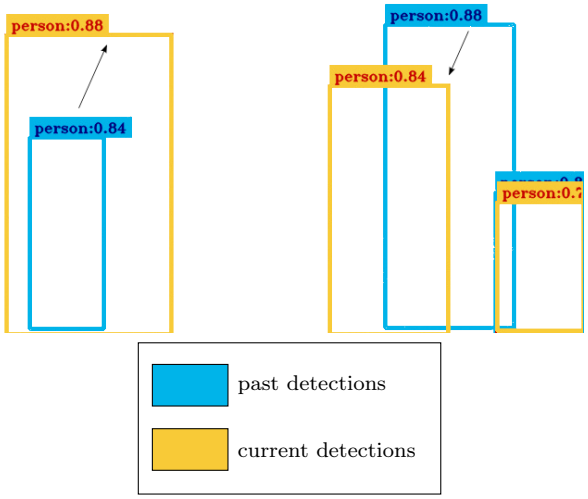



**Figure 8:** Detections matching

To solve this problem I used the Gale-Shapley algorithm [1] that always give a stable match with a quadratic complexity. In our case, the complexity is quadratic with respect to the number of past people detections that is small (even with a crowd of people we won't go higher than $n \approx 100$, ie $n^2 \approx 10000$ which is fairly small).

But the problem here if that our two sets of detections is not necesserally equally sized: what if someone appears or disappears in our new frame. To solve this, I made the previous algorithm stop whenever one of the two sets is exhausted. We are then left with one possibly non-empty set, if it is the past detections one, it means that the remaining people have disappeared, if it is the current detections one, it means that the remaining people have appeared.

Thanks to this algorithm we keep an environment of people that we see and we track them all from one frame to the other. Given the operator's detection at one frame, we can keep track of their position as long as there are people in our field of view under the assumption that those people don't move too much from one frame to the other. You can see the result of the algorithm on an example in Fig. 8.

However, this is not enough. Our assumption is too strong, we get the detections at an approxima-

tive rate of 5 FPS, it is low enough for a person to get lost in a crowd, even if the person stays visible. And more importantly, we did not address one of the most difficult problem of tracking: occlusions. What if someone passes in front of our subject ? Well, with this method if at some point one person is fully occluded by the other (which means that the people detector only sees one person in the frame instead of two), the person in front would become our new operator, no matter if they are the right one or not.

**Enhancement** Until now, the only parameter we took into account for tracking is the positions of our subjects with respect to the camera. To address the issue mentioned above, we decided to consider the movement of the subjects. Instead of using the position of the past detections to find a matching, we use a Kalman filters to estimate the next position of each subject given their previous positions. A Kalman filter is an algorithm that uses a series of measurements observed over time, containing statistical noise and other inaccuracies, and produces estimates of unknown variables (in our case, the 3D position of the person). We run the Gale-Shapley algorithm with these estimations, find the matching and correct each Kalman filter with the new position of the subject they are tracking. In addition to that, we added a lifespan to each person in our environment, this lifespan is set to some default value (typically 10 frames, ie. 2 seconds) each time the person is detected on the image, and decreased each time the person is not detected. Once it reaches 0, the person is considered lost and is deleted from the environment. When two people occlude each other in the RGB image, they have the same 2D projection but they don't have the same 3D position and more importantly they don't have the same trajectory. When we go from 2 detections on the image to one detection, the person that disappeared is not lost, it stays in the environment and we keep estimating their probable location with their Kalman filter until they reappear or until it is too late.

With this enhancement, we managed to have pretty good results in simple scenarios, with a fixed camera and a small number of people on screen (3 to 4 in our tests). But as soon as the camera starts moving, we get more noise in our images (especially the depth image) and the 3D positions that we measure are not as precise as they were.

### II.1.b. Election manager

Until now, the only information we used is the position of the detection in our world, which is basically what the depth camera gives us. To improve the result, we decided to use the RGB camera too: we try to use an SVM to recognize our operator amongst all the detections.

**Training set** At the very beginning of the challenge, we ask our operator to rotate in front of the robot so we can gather some positive samples. We require that there is no other person close to the robot in its field of view during that phase, to avoid any false positive (we typically gather 60 positive samples). Once it is done, we use some predefined samples that we gathered at our lab ourselves with the same camera as negative samples. Those negative samples are pictures of several people (6 different people) wearing different clothes (with different colors) under different light conditions (natural light, artificial light, different light intensity, different time of the day etc..). We pick a given amount of negative samples (typically twice the number of positive samples) randomly from those images.

**Feature descriptors** We wanted to use several feature descriptors to learn the operator: hue histograms are relevant in our case since the operator is not likely to change their clothes during the challenge but it is not robust enough since we don't know anything about the light condition throughout the challenge, so we decided to use SURF descriptors as well.

I tried to look into LOMO[2], a feature descriptor for person re-identification, I implemented the feature extraction (and some auxiliary functions) in python (I used the matlab code provided by the authors as a base-line) but I didn't have enough time to test it properly so I didn't use it for the competition.

**Election** Once we have our dataset, we train several SVMs, one per feature descriptor. If the feature extractor does not output a fixed-length vector, we have to find another way to represent the

data or else we cannot train the SVM. For SURF, we use the K-Mean algorithm to construct a bag of words with all the 64-dim vectors from the positive and negative set, and for each feature descriptor we compute the histogram of words (using K-Mean) which is a fixed-length vector (the size is the number of words, which is the number of clusters in K-Mean).

Once we have our SVMs, for each new detection we compute the prediction of each SVM, which is a 1 or a 0 (operator or not operator) and the final result is the sum of all the predictions. We call this the election manager because each SVM approves or not one candidate and our goal is to elect the most wanted one as operator.

The final result of the tracking module (see appendix .2.) is the position of the operator at each frame. The tracking pipeline is quite fast but we receive the frames at 5 FPS from the recognition module so the actual update speed is around 5 Hz.

### II.2. Results

Our method to solving the challenge is the following.

1. Detect the people you see, the closest person to the center of the frame (in front of the robot) is the operator
2. From now on, for each frame, run the stable marriage algorithm to keep track of the people we see. Each person has their own unique ID
3. Start learning phase, ask the operator to rotate and to move a little bit in front of the robot
4. While learning, detect the operator and save the detection as positive sample (we learn for a given amount of frame)
5. Train the SVMs
6. Start following, tell the operator you're ready to go
7. For each frame, after the environment have been updated, run the election manager on each detection
   - If no operator was found (the election manager answered 0 for every detection), don't change the operator ID
   - Else, change the operator ID to the ID of the positive detection that is the closest to the previous operator's detection
8. If the operator ID is still in our environment of people we track (we still see the operator), output their position and start again from step 7. Else, we lost the operator, say that you lost the operator and stop.

The main problem we had was that the operator and the person that was chosen to occlude the operator during the following part were moving very slowly. Way too slowly compared to a "natural way" of walking. Once the operator was occluded, the robot continued to follow the Kalman filter's predictions for some frames, but the lifespan of the detection ended before we could see him again (the occlusion lasted for several second, and the occluding person was moving perpendicularly to the operator). We didn't expect them to be this slow, so our parameters were not adapted to the situation.

## Conclusion

A total of 15 teams competed our league : the RoboCup@Home2017 Open Plateform League. Most of them (if not all except us) were not on their first trial. We tried to solve 3 challenges out of the 4 challenges in stage I and we ended up ranked 3 out of 15 in one of them (Storing groceries, see section I). We worked for three months on this project and


**Figure 9:** Best scientific poster certificate


**Figure 10:** Part of our team

our final ranking was 12 out of 15. We got the first place for the poster presentation (see appendix .3.) in our league and we won the best scientific poster award, all leagues mixed up.

Our code is open-source and available on ... The result of my internship is the perception module. A part of it (the recognition on an RGB image and the election server) is a standalone python library. It is easy to use in a different project: I used it for doing object and people recognition with the Hololens (augmented reality glasses by Microsoft).

## Acknowledgments

I would like to thank Dr. Pablo Lanillos for his expert advice and help throughout this difficult project, as well as Germán Díez Valencia for his wonderful collaboration. You supported me greatly and were always willing to help. Thanks to Patrick Grzywok, Emilka Skurzyńska and the rest of the team for their friendship in the lab. Finally, thanks to Dr. Karinne Ramírez Amaro for her supervision and her encouragement. This project would not have been impossible without the support of TUM, PAL Robotics, TNG and our other sponsors.

# References


[1] D. Gale and L. S. Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962.

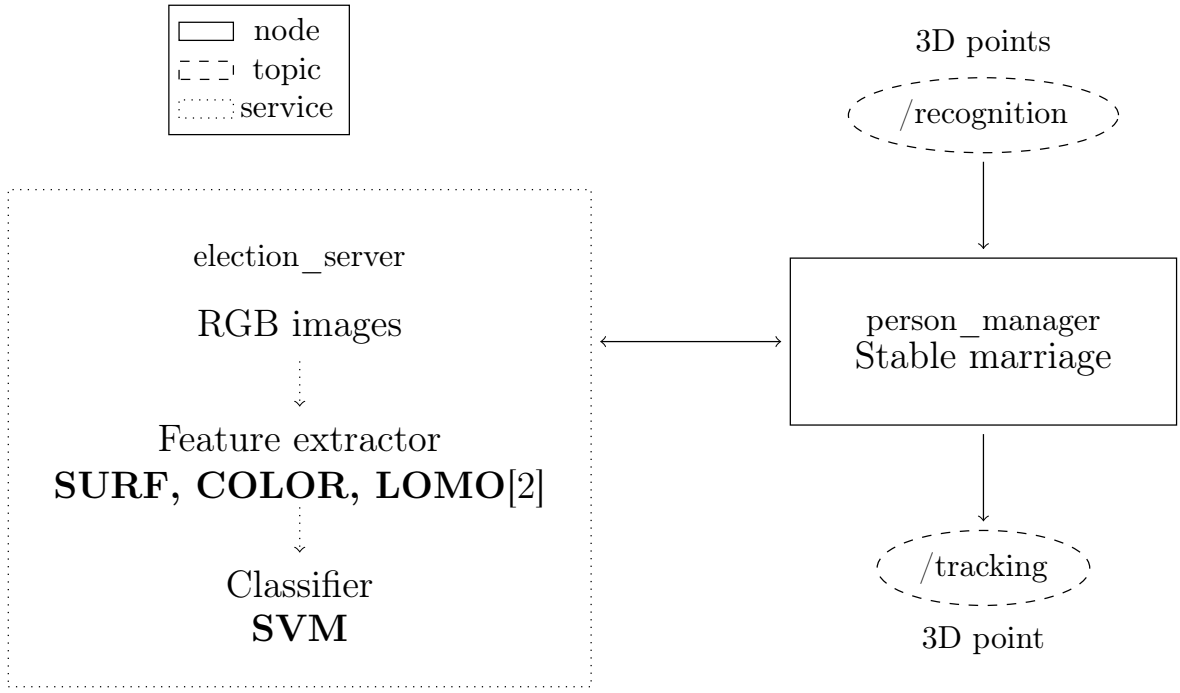
[2] S. Liao, Y. Hu, X. Zhu, and S. Z. Li. Person Re-identification by Local Maximal Occurrence Representation and Metric Learning. *ArXiv e-prints*, June 2014.

[3] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. *arXiv preprint arXiv:1612.08242*, 2016.

# Appendices

## .1. Recognition module

### .2. Tracker module



## .3. Poster