Object Recognition and People Tracking in a home environment

Ismail Lahkim Bennani

August 27, 2017

Contents

Introduction

Currently, object detection, recognition and tracking are three of the most important problems of computer vision.

Deep learning methods have astonishing results in object recognition but they require training and they cannot detect unknown objects whereas other methods without neural networks are not yet satisfying enough.

Concerning object tracking, we know some very accurate solutions for fixed cameras but we still have a lot of trouble when the camera is moving.

For my M1 internship, I was at ICS (Institute for Cognitive Systems) in Munich. I stayed there for 5 months and my main work was about object recognition and people tracking using a camera mounted on a robot's head. The robot we used was Tiago (see Fig. 1), it was lent by PAL Robotics to ICS therefore we could compete in the Robocup@Home2017[1] competition. The Robocup@Home competition is the largest international annual competition for autonomous service robots. It consists in several challenges that take place in arenas that are built like a normal apartment, with everything that could be found in an apartment.

My supervisor offered me to join a team of 3 master students, 2 PHD student and 2 researchers (that later became a team of 7 master students, 2 PHD students and 3 researchers) to work on the Robocup@Home project and I agreed, I've been working with them for 4 months and we went to

[1] http://www.robocupathome.org

Nagoya, Japan to compete. Before I joined the team, I worked on object recognition with the Hololens[2], a pair of augmented reality glasses made by Microsoft.

The result of my internship is the python code I wrote that will soon be released on github, it is basically a set of tools to do object detection, recognition and tracking using an RGB(D) camera. It relies a lot on the scikit-learn [8] and opencv [5] library.

First of all, I will present my work through two of the challenges we had to complete for the competition, then I will briefly talk about my work with the Hololens.

[2]https://www.microsoft.com/hololens

Figure 1: Our robot, Tiago

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 [10].

Setup The challenge takes place in a kitchen, there are at least a cupboard with some objects on it and a table with at most 10 objects on it. The robot starts facing the cupboard, the starting position is not specified, the table is less than 2 meters away from the cupboard. Their is one door on the cupboard, covering at most half of it. The objects on the cupboard and on the table can be "known" objects, "alike" objects or "unknown" objects. Some objects are given to the teams the day before the challenge, those are the "known" objects. The "alike" objects are objects that resembles "known" objects (for example, a bottle of coke and a bottle of water). The "unknown" objects are objects that are completely different.

Task The robot must start by opening the door, it can ask the referee to do it instead. Then it should look at the cupboard and identify the objects on it. Then it has to find the table, go to the table and identify the objects on it. Then it has to grasp on of the object and place it on the cupboard. It should allocate similar objects together (bottles should go with bottles on the cupboard) as much as possible. Then it has to go back to the table and start again, until there are no more objects on the table.

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[9] 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. 4)

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, therefore I decided to create a 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 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 therefore 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.

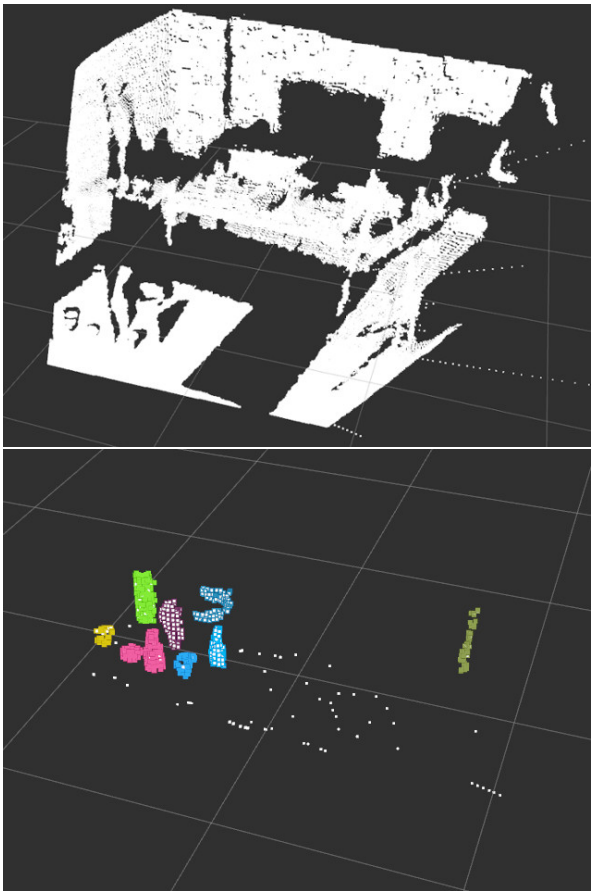


Figure 2: Point cloud (top) and object clusters on the table (bottom)

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 therefore 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 ie two points are clustered together if they are close enough to each other (see Fig. 2). The threshold used in Fig. 2 is 2cm, that means that two points are considered to be part of the same object (cluster) if they are less than 2cm apart from each other. 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. 4).

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: the rims of the shelf were unknown objects, some pieces of the wall were not big enough on the point cloud (because of partial occlusion of the wall by other objects) to be removed as big planes, therefore 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 therefore 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. The intersection over union (see Fig. 3) is an evaluation metric often used in object detection, here I use it to determine if two detections are the same or not. 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.

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

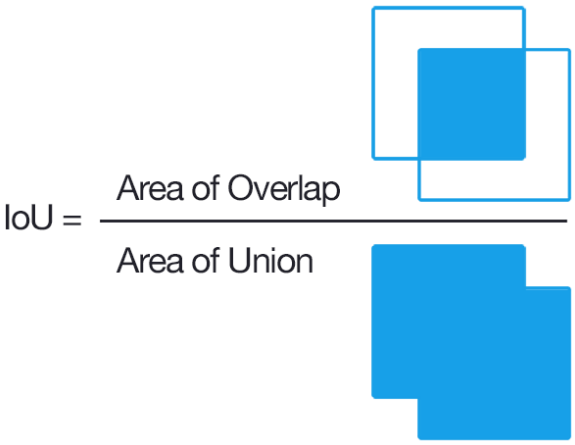


Figure 3: Intersection over Union (IoU)

Figure 4: YOLO predictions (top), predictions from depth clusters (middle) and the final predictions (bottom)

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 (see Fig. 6). 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 [7] developed by ICS[3]. The skin has several sensors such as a pressure sensor, a thermometer, a proximity sensor etc.. The robot moves the gripper toward the position given by the perception module and we use the proximity sensors on the skin cells to adjust the gripper's position and size to the object's actual position and size.

The final result of the recognition module (see Fig. 7) is the label, position and size of each recognized object relative to a fixed coordinate frame (see Fig. 8).

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.

[3]https://www.ics.ei.tum.de

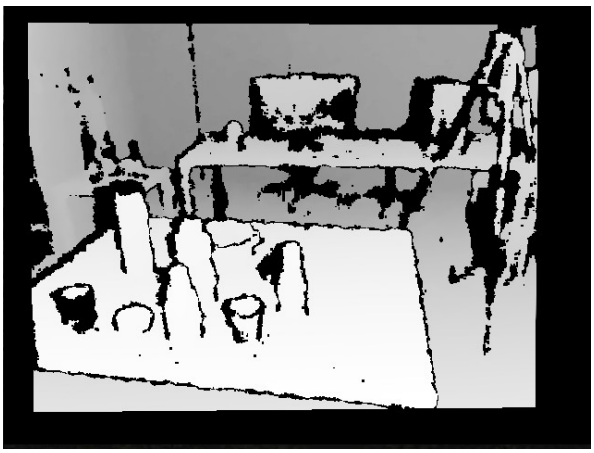


Figure 5: Depth image

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
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 the robot's head several times, see Fig. 9) 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.

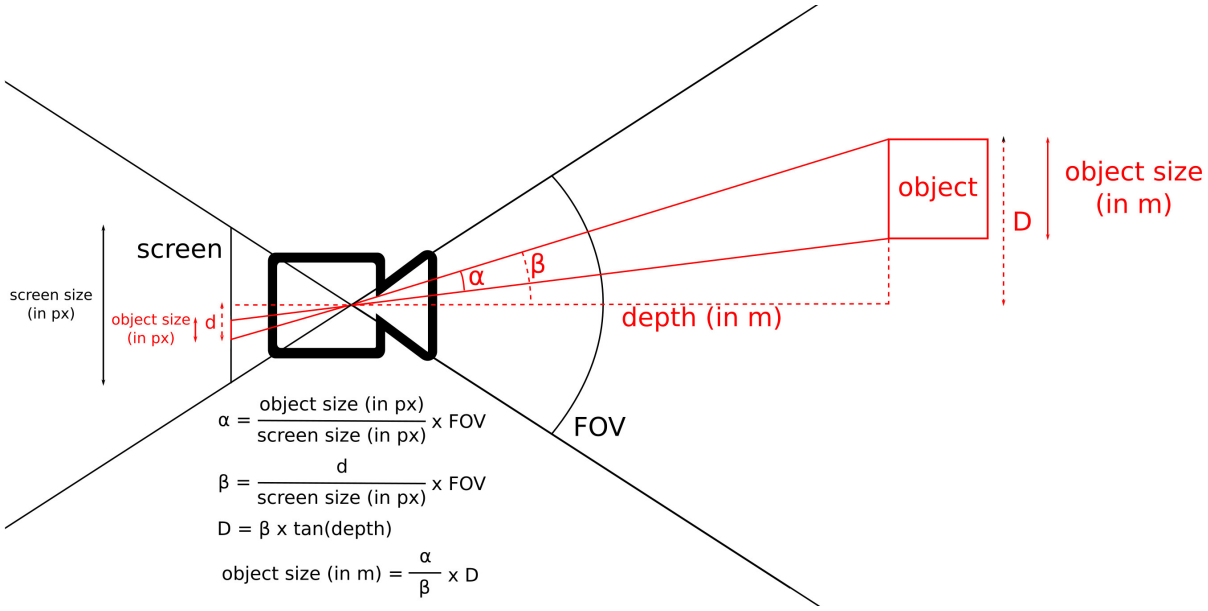


Figure 6: Compute the size of the object with the depth information

Figure 7: Recognition module

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 [10].

Setup The challenge takes place in the arena and outside of it, the robot starts waiting at a random location in the arena. There is a car somewhere outside the arena and several bags with groceries next to it. A professional operator is selected to act as the operator of the robot. There are no restrictions on other people walking by or standing around outside of the arena.

Task The robot should wait for the operator to say "follow me", then it should memorize the operator. The robot may ask the operator to perform some setup procedure. When the robot is ready, it should start following the operator who should start walking, in a natural way, towards the car. When they reach their goal, the operator will indicate the robot that they arrived, then they will give it a bag of groceries. The robot should bring the bag to a specific location inside the arena.

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

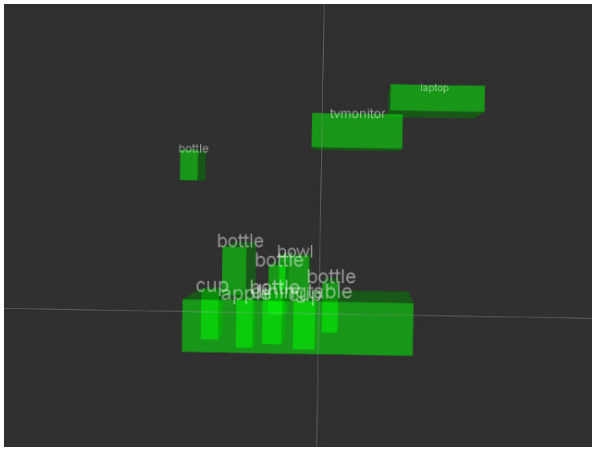


Figure 8: Recognition result

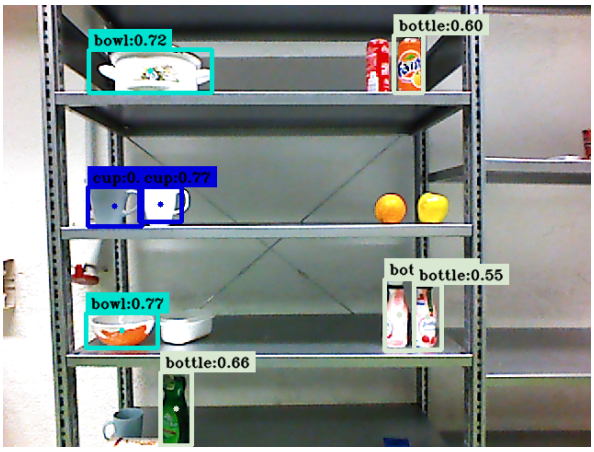


Figure 9: Shelf objects recognition

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 [3]. 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 [2].

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 euclidean distance between the position of the people we detect (that we get from the recognition module) to order the elements. The closer they are to each other, the more likely they correspond to the same person.

To solve this problem I used the Gale-Shapley algorithm [4] 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. 10. If we ignore the RGB image and only take into account the bounding boxes, the algorithm finds a very good solution.

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

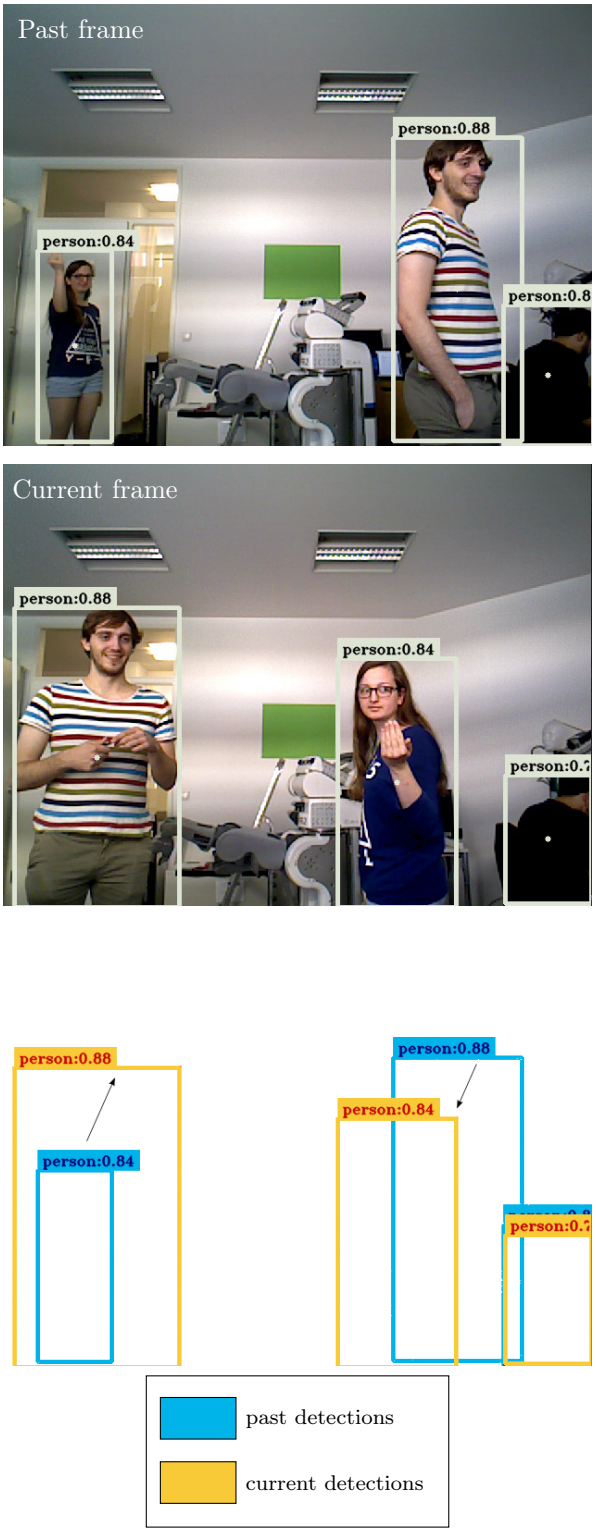


Figure 10: Detections matching

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. We can see on Fig. 10 that it is not enough, if the people move too much between two frames, our result are wrong. 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 therefore 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, therefore we decided to use SURF descriptors [1] as well.

I tried to look into LOMO[6], 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 therefore 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-NN 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 Fig. 11) 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 therefore 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, therefore our parameters were not adapted to the situation.

III. FORHD: Face and Object Recognition on a Holographic Device

FORHD is the name of the Hololens application I developed. I worked on this application before I started working on the Robocup@Home project and it was during this time that I came up with the architecture of the perception module. At the end of my internship, I updated the server-side of the application to make it use the new perception module, since I was using a very similar interface than before, it took me a fairly small amount of time.

The object recognition is based on YOLO, the same convolutional neural network used with Tiago. It needs a graphic card to run in real time therefore I had to use an external computer to do the computation. I decided to create a server on a desktop computer, the server would handle the object recognition part and will store an environment containing every object that we recognized. Then the Hololens would run a client that will send pictures to the server, compute the 3D position of the

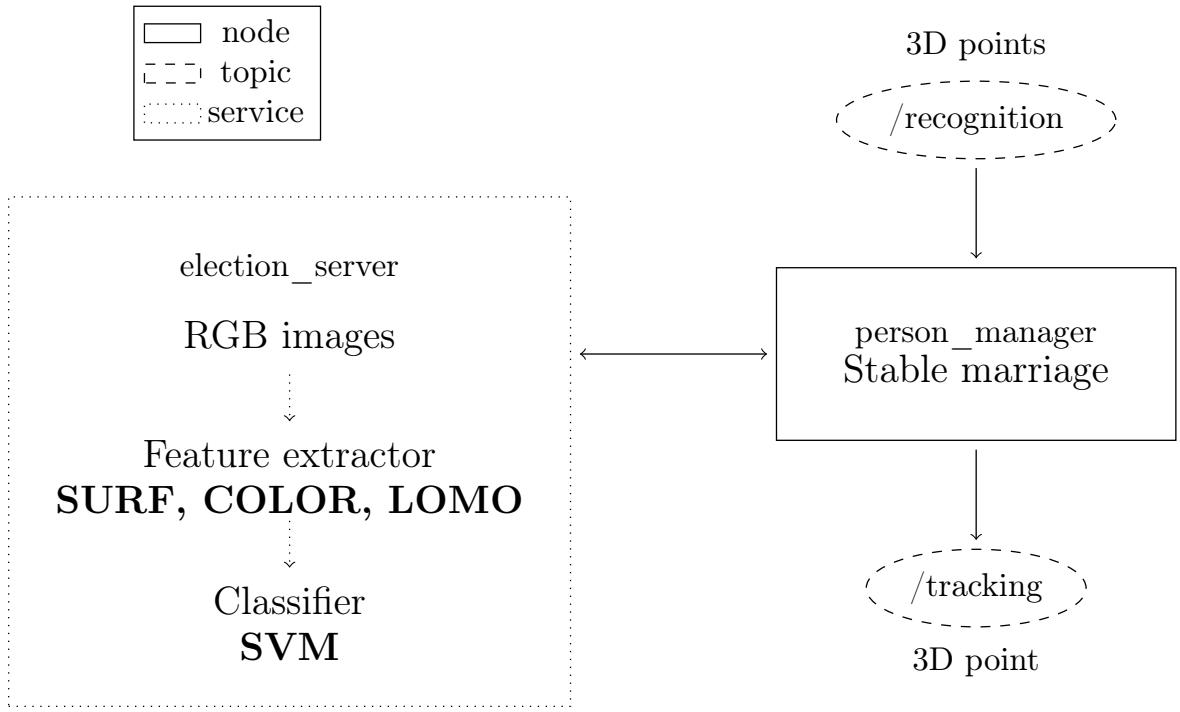


Figure 11: Tracking module

recognized objects and draw their bounding boxes as a hologram.

Client-side The client is the program running on the Hololens, it is the one that has access to the RGBD camera and that is interacting with the user. I wrote this program in unity, as advised by Microsoft. They have released a toolkit to help unity beginner to get started with the Hololens, that toolkit has, amongst other things, classes to handle the 3D representation of the room we are in and some examples on how to do raycasts with unity. Raycast is the tool I used to go from the 2D position of an object on the RGB image to its 3D position in the world.

Server-side The server is the program running on the desktop computer, it is the one that runs the perception module. It is an HTTP server waiting for requests. I created a small protocol for my application using GET, POST and PUT requests. For example, the client sends pictures and transformation matrices to the server with a PUT requests. I tried to do as much computation as I could on the computer since it is a lot more powerful than the Hololens.

The server is always waiting for requests, the only state it has is an environment of objects. The client main loop consists on sending pictures at a given rate. That rate is given by the server and is directly linked to the time it takes to process one image. Along with each picture, the client sends the current projection matrices (world coordinate frame to camera coordinate frame and camera coordinate frame to picture coordinate frame). For every picture it receives, the server will use the perception module to recognize objects on it, with the positions of the objects on the image and the projection matrices, it computes a 3D ray going from the camera position towards the object, the 3D ray of each recognized object is then sent back to the client. Then the client computes the position of the object by casting the 3D ray and looking for the first collision and sends the 3D positions (and size, we use the same method as seen in Fig. 6 to compute it) of all the objects to the server. The server updates its environment: if two points are

too close to each other and have the same label (the distance threshold depends on the size of the object) then it is probably the same object. Then it answers with the new environment, the client will then draw that new environment on screen.

The client-server approach allows us to have several client sharing informations. ICS recently bought 2 other pairs of Hololens, it would only require a small number of changes to the code to make the server work with several clients at once.

Conclusion

A total of 15 teams competed in 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 our final ranking was 12 out of 15. We got the first place for the poster presentation (see appendix .1.) 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).

Figure 12: Best scientific poster certificate

Figure 13: Part of our team

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] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Computer Vision and Image Understanding*, 110(3):346 – 359, 2008. Similarity Matching in Computer Vision and Multimedia.

[2] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, Sep 1995.

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

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

[5] Itseez. Open source computer vision library. `https://github.com/itseez/opencv`, 2015.

[6] 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.

[7] P. Mittendorfer and G. Cheng. Humanoid multi-modal tactile sensing modules. In *IEEE TRO - Special Issue on Robotic Sense of Touch*, pages 401–410, Jun 2011.

[8] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

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

[10] Loy van Beek, Dirk Holz, Mauricio Matamoros, Caleb Rascon, , and Sven Wachsmuth. Robocupathome 2017: Rules and regulations. `http://www.robocupathome.org/rules/2017\_rulebook.pdf`, 2017.

.1. Poster

Institute for Cognitive Systems
Department of Electrical and Computer Engineering
Technical University of Munich

Alle@Home team -Open Platform League, RoboCup@Home 2017

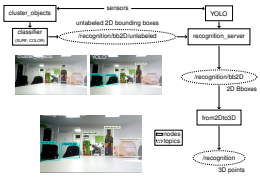
Karinne Ramirez-Amaro, Ismail Lahkim Bennani, Patrick Grzywok, Emilia Lozinska, Yumin Sun, German Diez Valencia, Matthias Humt, Ilya Dianov, Rogelio Guadarrama, Pablo Lanillos, Emmanuel Dean-Leon, and Gordon Cheng

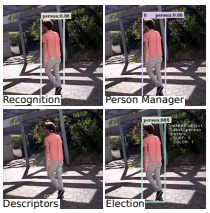
Abstract: We developed a system that integrates different capabilities such as reasoning & knowledge representations, object recognition, navigation, kinematic control, speech recognition and face detection. These capabilities allow the service robot TIAGo to perceive and recognize objects from different places, also to learn, recognize, and follow a new operator. The presented system is fully enhanced with our introduced reasoning and knowledge module to infer incomplete information from observations or verbal commands. We also present a novel control framework that considers information from our developed robot skin to improve the manipulation skills of our robot. Furthermore, we present a novel hierarchical approach that is capable of inferring on-line newly demonstrated everyday human activities from virtual scenarios.

Perception Module

Goal: Observe and **recognize objects** using RGB-D input to compute their 3D position in a real environment.

- Detect objects using YOLO [2] a **deep convolutional network**, which is **robust** recognizing multiple objects.
- Enhance the recognition of objects applying an **Euclidean clustering** method based on depth data.



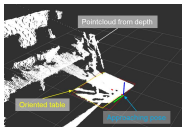


Goal: Learn, identify & **track the operator**.

- To update the environment of people, we treat that as a **stable marriage problem** using the Gale-Shapley algorithm [3].
- To detect the operator, we use RGB images trained with **hue histograms** and **SURF** descriptors.

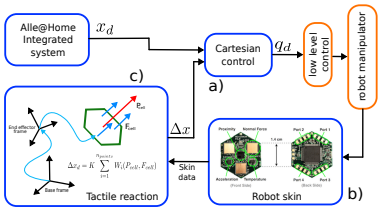
Goal: Identify **tables** in the environment.

- The RGB-D image is used to **segment planes** from the scene using RANSAC.
- A **convex-hull** is computed to obtain the boundaries of the plane.



Control Module

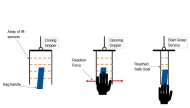
Goal: Design **controllers to manipulate** different objects from different locations, e.g. **grasping** from a table and **placing** on a shelf. We develop a library of low-level controllers to produce different robot behaviors applicable to different robots [5].



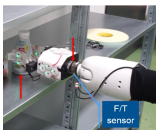
a) A cartesian controller tracks the desired positions and orientations commanded by the higher level nodes.

b) **Robot skin** is used to sense distance from objects (**Pi**) and contact forces (**Fi**) in the end effector in a n-cells network.

c) Skin signals Fi and Pi of Celli are used to compute the interaction forces, which are used to shift the desired cartesian position to make grasping and placing safer.



Interaction of the human hand and the gripper during bag handover.

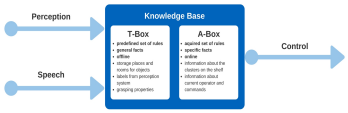


Examples of external forces applied to the system between the gripper, the object and the shelf.

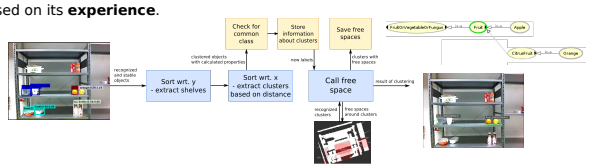
Reasoning and Knowledge Module

Goal: **Infer incomplete information** from several sources of information

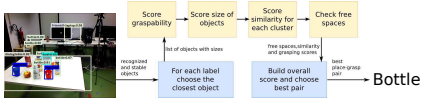
- Detect **relationships** between objects using general facts (**TBox**).
- Properly store the acquired knowledge from tasks (**ABox**) to **make informative decisions**.



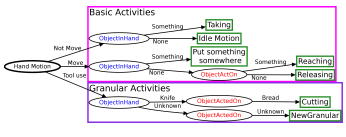
For example, we are able to **cluster** the recognized objects based on their **properties**. Then, the system makes a decision about the best graspable object based on its **experience**.

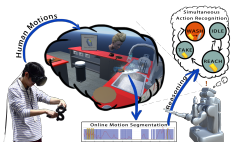


Retrieving information from the knowledge base to **infer the best object to grasp**.

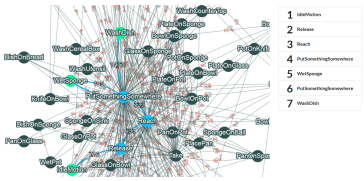


We present a **hierarchical approach** to extract the meaning of demonstrations by means of **symbolic and semantic representations** [1]. These **general common representations** are used to generate a semantic reasoning engine to **transfer** the obtained models among different domains [6]. Our reasoning-based learning system allows robots to **re-use their previous experiences** to correctly segment and **recognize newly** demonstrated activities for new tasks [1,4].



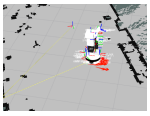


- A new method to teach robots **new tasks** in a fast and efficient manner by **extracting key structures** from the shown tasks [4].
- Automatically obtain the task relationships as a directed graph to **capture** important task components.



Navigation Module

We use the ROS navigation stack enhanced by the reasoning system. For building the map we use Karto and for the localization, we use the **Adaptive Monte Carlo Localization** approach with laser scanner data. We introduced 3 obstacle layers for sonars, lasers, and RGBD camera.



References

[1] Karinne Ramirez-Amaro, Michael Beetz, and Gordon Cheng. "Transferring skills to humanoid robots by extracting semantic representations from observations of human activities". Artificial Intelligence, Special Issue on {AI} and Robotics, Vol. 247, Pages: 95-118, 2017.
[2] Joseph Redmon and Ali Farhadi. "Yolo9000: Better, faster, stronger". arXiv preprint arXiv:1612.08242, 2016.
[3] D. Gale and L. S. Shapley. College admissions and the stability of marriage. The American Mathematical Monthly, 69(1):9–15, 1962.
[4] Tamas Bates, Karinne Ramirez-Amaro, Tetsunari Inamura, Gordon Cheng. "On-Line Simultaneous Learning and Recognition of Everyday Activities from Virtual Reality Performances". IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2017.
[5] Emmanuel Dean-Leon, Florian Bergner, Karinne Ramirez-Amaro and Gordon Cheng. "From Multi-modal Tactile Signals to a Compliant Control". IEEE-RAS International Conference on Humanoid Robots, 2016.
[6] Karinne Ramirez-Amaro, Emmanuel Dean-Leon, Ilya Dianov, Florian Bergner and Gordon Cheng. "General Recognition Models Capable of Integrating Multiple Sensors for Different Domains". IEEE-RAS International Conference on