A Hybrid SLAM and Object Recognition System for Pepper Robot

Kaisar Kushibar
kk33@hw.ac.uk

Songyou Peng
sp40@hw.ac.uk

Paola Ardón
paa2@hw.ac.uk

***Abstract*— Humanoid robots are playing increasingly important roles in real life tasks especially when it comes to indoor applications. Tasks such as indoor environment mapping, self-localization and object recognition help on making a robot system more autonomous, hence more human like. The well-known Aldebaran service robot Pepper is a suitable candidate for achieving these goals. In this paper, a hybrid system combining the object recognition with simultaneous localization and mapping (SLAM) is proposed for Pepper robot for the first time. The object recognition technique is based on SIFT and RANSAC which together prove to be a robust and efficient real-time technique for the system. The recognition is then integrated with a modified version of ORB SLAM 2 allowing Pepper to explore, map and recognize objects efficiently in an indoor environment. Due to the practical application of our system, it can easily be improved by applying path planning algorithms or object grasping techniques among others.**

***Index Terms*— Robotics, object recognition, SLAM, Pepper robot, ROS.**

I. INTRODUCTION

THE technological advance in the past decades significantly improved the quality of people's daily life. On an attempt of making humans lives easier and more independent, a great step has been made in the robotics and computer vision field – the development of humanoid robots.

In comparison with the rest of humanoid robots in the market, Pepper, developed by Aldebaran and Softbank, is affordable and has an open platform that allows developers to enhance its capabilities and implement applications to make the robot useful for every day task.

A. Problem Statement & Objective

The development of humanoid robots is still a relative new technological field, therefore a number of research is still being done in the subject. In order to achieve a higher level of interest and fresh ideas in the area many competitions are organized around the world. The European Robotics League (ERL) is an European-based common framework for robotics competitions, being Service Robots (ERL Service Robots) framework one of their focus. Thanks to the many advantages that Pepper robot offers, it is one of the robots used in the competition.

One of the purposes of the competition is to develop technological applications that will help elderly people to live longer independently at home [1]. Pepper comes with many built in functions, some of them being *learn home, object recognition* and *go to goal*. Even though these functions prove to be useful they have many limitations if the robot needs to be used for real life purposes. For example, the built-in function *learn-home* requires the environment to be less than $2m^2$. In order to cope with the rules of the ERL Service Robots competition and overcome the limitations of Pepper we set a main objective: to develop a hybrid system for Pepper that integrates the object recognition into Simultaneous Localization and Mapping (SLAM).

Another limitation that makes the project more challenging is poor sensors that comes with Pepper robot. Since both, visual SLAM and object recognition use optical cameras we are giving some characteristics of the used sensors [1]:

- *RGB camera:* located on the forehead and has a resolution of 320×240 at 5 frames per second (fps).
- *Depth camera:* Pepper has a depth camera located behind its eyes with a resolution of $320x240$ at 5 fps.

B. Contributions & Outline

Currently many applications have been developed on the robot family of Aldebaran and Softbank. However, to the authors acknowledge, **visual SLAM and its combination with a robust object recognition algorithm have never been done on Pepper robot before**. Our main contributions are as follows:

1) For the first time, a visual-SLAM algorithm is successfully applied in the Pepper robot so it is no longer just limited to a small environment.
2) Proposed an accurate and robust object recognition algorithm for Pepper.
3) Built a hybrid system which integrates a robust object recognition into the modified ORB SLAM. The recognized objects are marked in the map and the map can be saved and reused.

The paper's structure is as follows: Section II discusses many state-of-the-art object recognition and SLAM method. Section III introduces how the project was organized. The theory of our object recognition method and Simultaneous Localization and Mapping (SLAM) with Pepper robot are shown in Sections IV and V respectively. Then, the integration of the two functionalities into one system is described in Section VI, which is followed by the results (Section VII) and final remarks as well as future works in Section VIII.

II. STATE OF THE ART

Before going into details about the used algorithms for the implementation it is useful to review some of the general concepts in the field. Object recognition and SLAM algorithms have been active research fields over the last 20 years. Some of the related work is reviewed in this section.

Source code of the project is available in the git repository: **https://github.com/PaolaArdon/Salt-Pepper**

A. Object Recognition

Object recognition relates to the problem of identifying an object in an image. In general, the algorithms can be divided in two main streams: appearance-based methods and feature-based methods.

An **appearance-based** recognition method is based on directly using example images (or templates) to perform recognition tasks. One of the methods in this branch is based on edge and frame extraction described by Sung *et. al* in [2]. These are then matched with templates in a database using sliding windows and the ones with high similarity are considered to be the recognized object.

Another type of appearance-based methods is focused on the histogram. Swain & Ballard in [3] initially showed how the object recognition can be performed by comparing the RGB color histogram. Schiele & Crowley [4] applied histograms of receptive fields proposed by Koenderink & van Doorn [5] and the recognition result is enhanced with the usage of the Gaussian derivative or the Laplacian operator at multiple scales. Linde & Lindeberg [6] generalized the idea of the receptive field histogram to higher dimensionality, which significantly improves the recognition performance. In the paper of Schneiderman & Kanade [7], histograms of wavelet coefficients are proved to be an useful tool for the recognition of cars and faces.

Appearance-based methods are usually robust to a certain type of object characteristics, depending on which information is extracted for the comparison between the templates and the objective image. However, it turns out most of them are sensitive to many variations and are computationally expensive. On the other hand, feature-based recognition methods offer a solution to most of the previous mentioned problems which makes them a popular choice.

The objective of **feature-based** recognition algorithms is to find feasible matches between the features extracted from the database images and from the target image. Some of the commonly used features in object recognition are:

Shape Context*:* a descriptor proposed by Belongie *et al.* [8] to measure the similarity between two shapes to be used for object recognition. The shape context is obtained for each pixel and for the one at a reference pixel, which captures the distribution of all the remaining points within the same shape. Even though in theory the method has proven to work well for objects with distinguishable shapes, this is not always the case in real world applications.

Haar-Like feature*:* Haar-Like features are used in Haar cascade algorithm, which used for many applications like face recognition; locate pedestrians, objects or facial expressions in an image. This type of feature is defined within a small window, and sums up the difference of the pixel values in the adjacent rectangular regions In general, for Haar cascade the system is provided with several training images that give the needed features to be used later in a classifier [9]. The disadvantages of this technique are mainly the long training time and lack of robustness, which will be further compared in Section VII.

SIFT*:* SIFT is a tool to detect the keypoints of an image using Difference of Gaussians (DoG), describing them through the histogram of weighted orientations. SIFT is invariant to many factors, such as rotations and scales, among others. More details will be discussed in Section IV-A.

SURF*:* The Speed Up Robust Feature (SURF) algorithm is a scale and rotation invariant point detector and descriptor [10]. SURF uses an approximated DoG and the integral of the image. After an image is converted into an integral image, we can compute the block subtraction between any two blocks with just six calculations. Therefore, computing SURF is theoretically three times faster than computing SIFT. However, the SURF descriptor is based on the distribution of first order Haar wavelet responses. From our experiment during the usage of Haar the performance of the descriptor is not as outstanding as the one of SIFT.

ORB*:* It is subject to the FAST keypoint detector and the visual descriptor BRIEF. More details about ORB can be found in Section V-A.

BRISK*:* Binary Robust Invariant Scalable Keypoints (BRISK) also consists of the detection and descriptor part [11]. For the detection, a scale space is created and the scored used in FAST detector is computed across the space. Then, the pixel level non-maximal suppression is performed. To describe the detected keypoints, the weighted Gaussian average over a select pattern of points near the keypoints are calculated [12]. The Hamming distance instead of Euclidean is used to match the keypoints, which helps to speed up the process.

The comparisons of the recognition performance using various features will be detailed in the result & discussion part (Section VII).

B. Simultaneous Localization and Mapping (SLAM)

In this section we are going to review of some of the state of the art algorithms for SLAM that we have considered in our project.

Extended Kalman Filter SLAM (EKF-SLAM)*:* In EKF-SLAM the map is represented with a large vector stacking sensors and landmarks states which is modelled by a Gaussian variable [13]. Maximum likelihood algorithm is used for data association.

Some of the advantages of EKF-SLAM is that is relatively easy to implement and is efficient when working with small number of features and distinct landmarks. On the other hand, the complexity is quadratic with respect to the number of features, it does not guarantee convergence in non-linear and/or non-gaussian cases, and does not correct erroneous data association [13].

Collaborative Visual SLAM (CoSLAM)*:* This algorithm [14] interacts in dynamic environments where live frames come from multiple cameras that can be independent and are mounted on different points of view. As an overview, these cameras build a single global map, including the static background points and the foreground dynamic points. This set of points are the ones used to estimate the poses of all cameras, which views should overlap. CoSLAM is

considered as one of the most efficient approaches which is able to get rid of false points caused by incorrect matching.

Large Scale Direct Monocular SLAM (LSD-SLAM): It has been developed to allow the building of large scale map environments [15]. In this SLAM version instead of using keypoints it uses the image intensities to track and map. The method shows the advantage of allowing the mapping of large areas without extra computational power.

Oriented FAST and Rotated BRIEF SLAM (ORB-SLAM): It is a keyframe and ORB based SLAM algorithm [16]. One of its greatest advantages is that it operates in real-time and large environments, being also able to close loops and relocalize from different viewpoints. Due to these great contributions it is the chosen algorithm for the project implementation. More details about it are described in section V

III. PROJECT MANAGEMENT

In order to accomplish the objectives we set a schedule for the project which we closely followed. We were given a total of nine weeks to accomplish the goals for the project.The detailed calendar is shown in the table.

Scheduling of the Project			
Task name	**Start**	**End**	**Total days**
Algorithms review	Sep 26	Oct 03	7
Check point #1	Oct 10	Oct 10	1
Visual SLAM	Oct 03	Oct 23	20
Object recognition	Oct 11	Nov 01	20
Check point #2	Nov 01	Nov 01	1
System Integration	Nov 04	Dec 04	28
Final Presentation	Nov 22	Nov 22	1
Final Demonstration	Dec 01	Dec 01	1
Report	Dec 12	Dec 17	5

As seen on the previous schedule, most of the project's time was spent on building the hybrid system which integrated the object recognition with SLAM. Furthermore, working with hardware can bring additional difficulties on the side.

One of the project management challenges is to overcome delays and still being able to deliver the expected outcome. Our case was not the exception, we had many delays on the way, such as: the 3-week late arrival of the Pepper robot, the allocation of a working space (lab allocation), and network problems among others. Despite of the faced difficulties, the system integration was successfully carried out.

IV. PROPOSED OBJECT RECOGNITION

As previously explained, We want to make Pepper more human like and helpful in the household. One of our main objectives is to allow Pepper to recognize objects. Some of the classical object recognition algorithms show not to be efficient for some applications. For instance, Haar Cascades [9] requires a trade off to be done between the efficiency in learning/training time and the output's accuracy.

Based on Lowe's paper [17], we proposed a robust SIFT-based recognition algorithm for Pepper robot. Our method is not only able to get rid of the long training time, but is also robust to rotation, scaling, perspective transformation among others. The robustness of this algorithm allows Pepper to recognize objects efficiently.

The flow chart of the proposed algorithm for Pepper is presented in Fig. 1. The method's main steps are:

- Feature extraction
- Feature matching
- Decision making

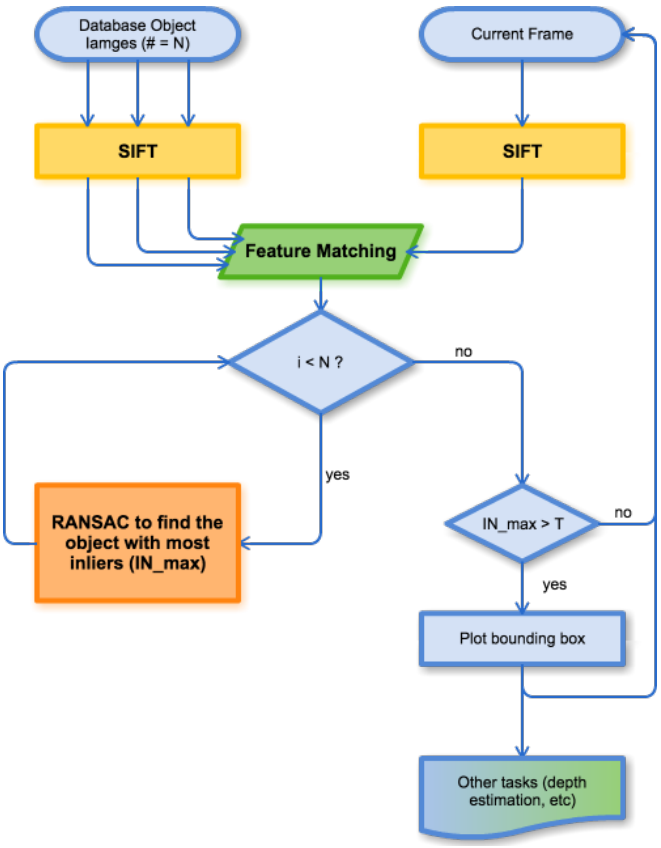


Fig. 1. Workflow of our proposed object recognition algorithm

A. Feature extraction

The very first step is to extract a feature from a given image database. This is done every time a new frame arrives. And every time when a new frame arrives, the first step is also to extract the features. Under the uniform recognition framework, we tested several feature extraction techniques including ORB, SURF and SIFT.

When applying ORB and SURF the accuracy detection of the output is acceptable. However, SIFT offers the highest accuracy of them all out of our testing rounds as we can see in Section VII, so we decided to choose SIFT as our primal feature extraction method.

However, we still keep both SURF and ORB as user options in the implementation. Although we will only discuss

SIFT for the rest of the paper, users can still easily change the feature to others from our code.

The SIFT process consists of two parts: keypoint detection and description. In the detection section, a scale space is constructed, based on which the Laplacian of Gaussians (LoG) is acquired to extract the keypoints. After a few post processing steps like eliminating the keypoints with low contrast, the keypoints are localized and the corresponding orientations are assigned.

For the descriptor part, the histogram of weighted orientations around the keypoint area is obtained, which represents the important information of the current keypoint. It has been shown that the resulting descriptor is invariant to the following: scale, rotation, additive illumination, perspective transformation and noise. This invariance is what allows SIFT to be a powerful descriptor and play an essential role in our task.

B. Feature matching

Once the SIFT features of the images in the database and the current frame have been extracted, we need to know how many features are a match between the current frame and every image in the database. This will help the following decision-making step described in the next section.

At first, we applied a brute-force matching scheme but soon replaced with a KD-tree based matching method which significantly speeded up the matching process.

Brute-Force matching: The easiest yet inefficient approach to feature matching is brute-force nearest neighbour matching (Alg. 1). The concept is as follows: first, compare each feature f_i in the current frame with all the features g of one certain database image, and find the one g_{min} with the shortest distance. Second, as a double check, the distances between g_{min} and all features f in the current frame are calculated and then the f_t with the shortest distance is chosen. If the f_t is exactly the f_i, we find one good match, otherwise we go to f_{i+1} and repeat the same process until all the features have been traversed.

Such a matching process for one single database image is described in Alg. 1. Then, the process should be applied for each object image in the database in order to find the right matches.

Algorithm 1 Brute-Force matching algorithm

Input: $F = [f_1, ..., f_n]$ features of the current frame, $G = [g_1, ..., g_m]$ features of one database image
  for $i = 1, ..., n$ **do**
    f_i finds the feature g_{min} in G with shortest distance
    g_{min} finds the feature f_t with shortest distance
    if $i = t$ **then**
      Add i to the best match index
    end if
  end for
Output: Indexes of the best matching features

Certainly, this matching method is quite computationally expensive. When the number of objects in the database is large, the calculation time will be increased drastically. In our experiment, when there are only three objects in the database, the computational time goes to more than 300 milliseconds to run the recognition for one frame.

If we remember from Section I the frame rate of the RGB $2D$ camera of Pepper is 5 fps, which means the frame is updated every 200 milliseconds. From the experiments, the longer we ran the recognition code with the brute-force matching, the more the frame updating time was delayed. In the testing rounds, for example: after running the recognition for some time, we moved an object into Pepper's camera field of view and the object appeared on the screen more than 20 seconds later.

Kd-Tree matching: Due to the inefficiency in the previous presented method we decided to use the matching method proposed by Beis *et al.* [18], which is a kd-tree built for the rapid traversing of each feature in the current frame. The main idea is as follows[19]:

1) Build a KD-tree for the features extracted from the database images by bisecting search space on the dimension with the greatest variance.
2) A feature f_i in the current frame traverses the constructed kd-tree and the corresponding distances between the feature and each branching point are saved in a queue.
3) Back-trace from bottom to top and check if the current leaf distance is less than all the distances from the queue. If yes, the non-visited branches will be traversed.

From our intensive experiments it has been shown that the kd-tree nearest neighbour matching algorithm significantly speeds up the recognition process (around three times faster than the one shown with brute-force matching). Since the recognition runtime for each frame is within the range of the updating time ($200\ ms$), the kd-tree matching enables Pepper to real-time recognition.

C. Decision making

Once the good feature matches between the current frame and all the objects in the database, the Random Sample Consensus (RANSAC) [20] is applied. RANSAC decides whether an object from the database exists in the current frame or not and which object it is.

As seen in Alg. 2, we use RANSAC to fit a homography transformation between the feature positions $M_C^{(i)}$ in the current frame and the object in the database $M_D^{(i)}$, and then calculate the number of inliers $N^{(i)}$. If $N^{(i)}$ is larger than a threshold (we set it to be 15) and also larger than the previous number of inliers, we replace the object i as the best candidate. This process is repeated until the number of inliers of all database objects has been calculated. If the inlier numbers are all smaller than the threshold, it is reasonable to say there is no object in the current frame, otherwise, the object with most inliers is considered as the detected object. Finally, if an object is detected, through the homography matrix that we have acquired from RANSAC, the bounding box of the best object position is drawn on the frame.

It is worth mentioning that when the threshold for the number of inliers is set properly, the object can be found despite of the obstructions between the object and the camera, section VII shows the results of the method.

So far, we have seen the algorithm process for one frame, once the process finishes it runs iteratively for each frame.

Algorithm 2 Decision making through RANSAC

Input: Matching indexes of the features in the current frame $M_C^{(i)}$ and one certain database image $M_D^{(i)}$, number of database image n, minimum number of inliers N_{min}, bestInliers = 0, bestIndex = 0
  for $i = 1, ..., n$ **do**
    $N^{(i)}$ = findHomography($M_C^{(i)}$, $M_D^{(i)}$, RANSAC)
    if $N^{(i)} > N_{min}$ && $N^{(i)} >$ bestInliers **then**
      bestInliers = $N^{(i)}$
      bestIndex = i
    end if
  end for
  if bestIndex = 0 **then**
    No object has been found in the current frame.
  end if
Output: bestIndex (the index of the best matching object)

More implementation details about the Pepper object recognition can be found in the `/pepper_recog` folder of our Github repository.

V. SIMULTANEOUS LOCALIZATION AND MAPPING

Nowadays Simultaneous Localization and Mapping (SLAM) is one of the active research topics in Computer Vision and Robotics community. Many SLAM algorithms have been developed as we have discussed in the Section II. All of these algorithms share a common purpose, however, they use different approaches depending on the available sensors. Some of them use laser, cameras or RGB-D cameras (or a combination of different sensors). For instance, LSD and ORB SLAM algorithms are based on RGB(-D) camera and we can call them visual-SLAM.

The needed sensors of Pepper robot have been described earlier, the ORB SLAM is the one that better copes with the sensors capabilities, therefore being the implemented algorithm on Pepper. In this section a brief introduction for ORB features and ORB SLAM is given as well as its extension ORB SLAM 2 [21] that uses RGB-D camera.

A. ORB feature

Since we are working with the visual SLAM, extracting features from the input video stream is commonly the essential step. Feature extraction is the base for ORB SLAM algorithm that makes the robot understand the surrounding environment and localize itself, as well as closing the trajectory loop. The Oriented FAST and Rotated BRIEF feature [22], known as the ORB, is a state-of-the-art feature that is applied to our SLAM algorithm.

ORB is built on the Features from Accelerated Segment Test (FAST) detector [23] and Binary Robust Independent Elementary Features (BRIEF) descriptor [24]. Original FAST detector does not provide neither the keypoint orientation, nor the measure of the cornerness, which makes ORB not rotation invariant. Therefore, in the phase of keypoint detection of ORB, the intensity centroid [25] and the Harris corner measure [26] are applied to remedy these disadvantages. Similarly, although the BRIEF descriptor can be calculated efficiently and robust to additive illumination change, perspective distortion, etc., the performance of BRIEF diminishes significantly for the rotation over a few degrees. To solve the weakness of BRIEF, the best BRIEF pairs with large variance and low correlation are learned from PASCAL VOC 2006[27] and then the obtained BRIEF descriptors from the keypoints of the current image are steered based on the orientation of the keypoints.

ORB is made up of the modified version of FAST and BRIEF that we mentioned before. It is rotational and scale invariant as well as robust to noise, and it has been shown that the performance of ORB in many real-life applications is equivalent to or even slightly better than SIFT in some cases. More importantly ORB is computationally inexpensive. Compared with the costly SIFT, ORB is at two orders of magnitude faster, which is suitable for our real-time SLAM application.

B. ORB SLAM – Monocular

ORB SLAM mainly consists of three components that run in parallel: *tracking*, *local mapping* and *loop closing* as shown in Fig. 2 In the following sections, the main idea of each component is detailed.

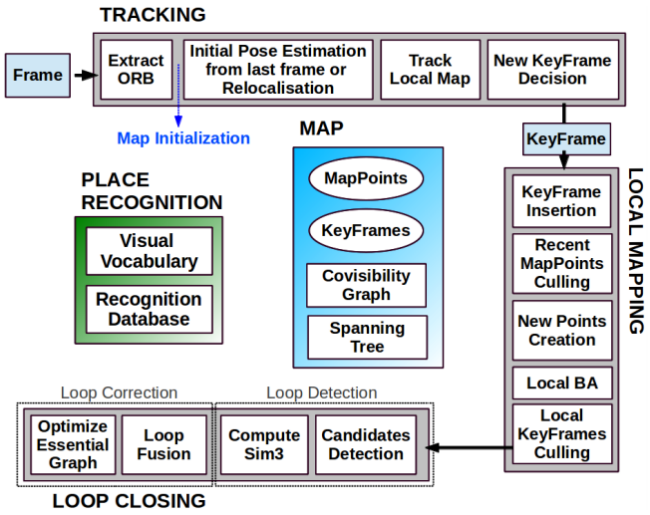


Fig. 2. Overview of ORB SLAM. Image courtesy of [16]

1) Tracking: This process starts with the *initialization* of the map. In the monocular case the depth has to be computed using several images of the same scene by moving the camera in horizontal/vertical direction with respect to the scene. The authors of the algorithm proposed a new method

[16] for "structure from motion" estimation that combines two geometrical models for camera pose estimation:

- Assumes the scene is planar and computes the corresponding homography matrix between two frames.
- Assumes the scene is non-planar and computes the fundamental matrix.

Then the selection of the best model is computed using a certain heuristics, and the camera pose will be estimated based on the selected model.

Once the map has been initialized from several consecutive frames of a scene from different viewpoints, the ORB features (key-points) are extracted from consecutive frames. It should be noted that the FAST corners are extracted at 8-scale levels and the modified BRIEF descriptors are computed on the keypoints orientation.

The camera pose is computed by searching the matches in a small area around each ORB keypoint between the current frame and the previous one. The search is optimized by assuming that the camera motion has a constant velocity model. If there are not enough matches, the search is done on all map points near the points from the last observed frame. In case the track is lost, the current keypoints are converted into bag-of-words features and traverse the predefined recognition bag-of-words database. This is applied to obtain the best matching keyframe. After that, the robot can be re-localized again. Moreover, PnP algorithm[28] along with RANSAC are applied to further refine the estimation of pose.

2) Local Mapping: From what we discussed in the last section, a new keyframe is obtained. In order to put the new map points, we need to find the positions of all the new points on the world coordinate. Instead of triangulating points only with the closest keyframes like PTAM, ORB-SLAM triangulates points with several neighbouring keyframes. As long as a pair of ORB features have been matched, they can be triangulated.

Sometimes wrong map points may appear. To ensure all the mapped points are the real ones, we should check if a map point is remained in the map for a period of time. The author of ORB-SLAM uses a method called *pass culling test*, which basically means a keypoint can be put in the map only after the following two conditions are satisfied: make sure the keypoint can be found in at least 25% of frames, and the keypoint should be seen in at least three keyframes.

Finally,the local bundle adjustment will optimize the current keyframe. The final pose optimization is performed by Levenburg-Marquart method.

3) Loop Closing: Loop closing is one of the most important contributions of the ORB-SLAM and also one of the reasons we chose it for Pepper's SLAM task. Loop closing means when the robot is moving around the environment and then comes back to the starting point, the system should be able to connect the latest movement with the initial ones. In this case, the trajectory can be closed and the map will be globally changed. With the loop closing, the built map and the estimated robot trajectory are more accurate.

The main idea of loop closing can be summarized in three steps: loop detection, similarity transformation computing and loop fusion. First,a co-visibility consistency test is performed to check if a loop has been found. Throughout the whole process of the SLAM, we keep calculating the similarity between the current keyframe and all its neighbors in the co-visibility graph. The keyframe with the highest similarity score will be used to update the reference loop-closing frame. Second, if one keyframe satisfies the test in the first step, the RANSAC will iteratively be applied to calculate a similarity transformation containing: 3 translations, 3 rotations and 1 scaling parameters. When the candidate has enough number of inliers, we are sure the loop has been found. Third, with the similarity transformation matrix acquired from the last step, the map points in the current keyframe is reformed to the reference loop-closing keyframe. The map points from all the neighbours of the current keyframe are also projected through the same transform. Therefore, all the inliers from the last step are fused.

The last step is to perform a global bundle adjustment. The only difference from section V-B.2 is that optimizing all the map points will be used for the bundle adjustment and refined.

The illustrations of the loop closing can be found in the Section VII.

C. ORB SLAM – RGB-D

As it has been mentioned, the first step of ORB SLAM is the initialization of the map, which requires several images of a scene from different viewpoints. However, this process takes a long time for Pepper robot, because with a rate of 5 fps the sequence of images cannot provide smooth parallax effect.

In order to overcome this problem, an extension for Monocular ORB SLAM has been introduced in [21], where the depth estimation has been replaced by the RGB-D camera. In this case, the initialization process does not involve recovering the camera pose from several images. Instead, the first taken image by the camera can be directly used to initialize the map because the depth information for the keypoints is already there. Therefore, using an RGB-D camera speeds up significantly the initialization process, which is very important when using a camera with a low frame rate as in Pepper robot.

VI. INTEGRATION & ARCHITECTURE

Our final objective was to combine the object recognition with SLAM, i.e. while running SLAM we also want the robot to identify the detected object's position and put a marker with label on the map.

In this section we are going to show how we accomplished this task. Also, how the whole system is organized in order to make the robot, ROS and the two previously described algorithms work together. The additional features integrated in the system will be introduced as well.

A. System overview

Pepper robot comes with many built-in functions and its own operating system (OS) called NAOqi-OS. This is a

Linux distribution based on Gentoo and it is installed in Pepper's computer which is integrated on the robot. However, Pepper doesn't allow to install third-party applications on its OS and requires to use its own Software Development Toolkit (SDK). In order to overcome this limitation, the Robotic Operating System (ROS) has been used in this project. ROS is a language and platform independent framework that allows users to create packages in a graph-based structure and provides a powerful tool for message sending/receiving between processes [29].

ROS – NAOqi Driver and plugin for Pepper: In spite of the fact that the manufacturers of Pepper limit the access to the OS of Pepper, they provide a driver that can be used to link NAOqi and ROS together. This driver fetches all sensor data and creates ROS nodes and topics which publish the states of all the robot sensors. Moreover, the driver creates topics for controlling joints of the robot allowing other ROS applications to subscribe and publish standard ROS messages (e.g. Twist) to control the robot. The whole process of NAOqi-ROS communication is illustrated in Fig. 3. As it can be seen from this figure, the main role of the NAOqi driver is converting NAOqi modules to ROS nodes.

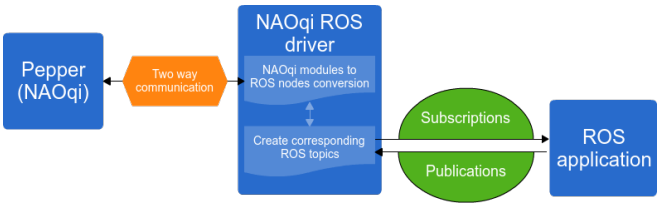


Fig. 3. The diagram illustrating the way of communication of a ROS application through NAOqi ROS driver.

In addition to the NAOqi driver, there must be robot specific plugins that bring specific capabilities of the robot to ROS depending on the characteristics of the robot. For example, Pepper robot shares the same OS with the other Aldebaran and Softbank robots. However, each of these robots have different characteristics such as number of joints and different types of sensors. In order to avail full robot capabilities it is required to run a certain type of driver. To achieve this, *pepper_bringup* and *pepper_dcm_bringup* plugins [30] have been used for Pepper robot. The main difference between pepper_bringup and pepper_dcm_bringup is that the former does not block the autonomous life of the robot whereas the latter turns that functionality off (which is what we mainly use). Where the autonomous life is an out of the box functionality that imitates human behaviour in the robot (e.g. track human face, react to a sudden loud noise, etc.).

B. Implementation

Now we introduce how SLAM and object recognition are combined using ROS. First of all we have to mention that the ORB SLAM 2 algorithm that we used has been implemented in C++ programming language as a *stand-alone* application, i.e. it can be used without ROS.For this reason, it does not use *RViz* for showing the map, which is a default and convenient visualization tool of ROS. Instead of RViz it uses *Pangolin* [31], which is a lightweight library for managing visualization and user interaction that wraps OpenGL library [32] functions.

1) ***Combining SLAM with ROS:*** To use ORB SLAM 2 in ROS, a ROS node was implemented to instantiate ORB SLAM 2 as an object. The created node subscribes to the topics where RGB and depth images are being published. Note that the ORB SLAM 2 with RGB-D expects the camera to be RGB-D, but Pepper has RGB and depth cameras separately. Accordingly, we made two separated subscribers for both modalities. We also have to make sure that the messages coming from these topics have the same *timestamp*, because it is possible that some frames may be delayed or lost due to unexpected technical issues. Furthermore we also make sure that the images from both cameras are correctly registered. The described architecture for SLAM & ROS is illustrated in Fig. 6 (Block-A).

2) ***Combining object recognition with ROS:*** In contrast to the ORB SLAM implementation, we implemented object recognition module as a ROS Node, so the algorithm logic (as illustrated in Fig. 1 is directly put inside the node. Then, we obtain images from Pepper's frontal camera and convert ROS raw image format to OpenCV image using CV-Bridge [33] package from ROS.

The object position with the respective camera coordinate (depth estimation) is computed in this node as well. The estimated depth information is published as a topic. For publishing the object name and its position in camera frame we created a custom ROS message that holds the following fields: 1) flag (boolean type) – accepts *true* when an object has been detected, *false* otherwise; 2) depth (float type) – estimated distance from the camera frame origin to the object; 3) name (string type) – name of the object that has been detected.

3) ***Depth estimation:*** Provided an object is recognized in the present frame, there are two following tasks that require the knowledge of the depth distance between the Robot and the recognized object. One is the object following and avoiding. Based on the depth information, Pepper can decide to move towards or backwards the object. The other task is to mark the objects' position on the 3D map while SLAM is being performed. Due to the fact that the depth camera of Pepper has severe distortions because of the lens covering out of the depth camera, it is essential to find a way to estimate an accurate depth distance without the depth camera.

A pinhole model is employed for estimating the depth information.

From Fig. 4, we can easily get the depth from the pinhole model:

$$d \approx f \times \frac{R}{r} - f$$

where f is the focal length, R is the real size of the object and r is the object size on the image plane. For the RGB camera of Pepper, we found the pixel conversion to millimetre is 1

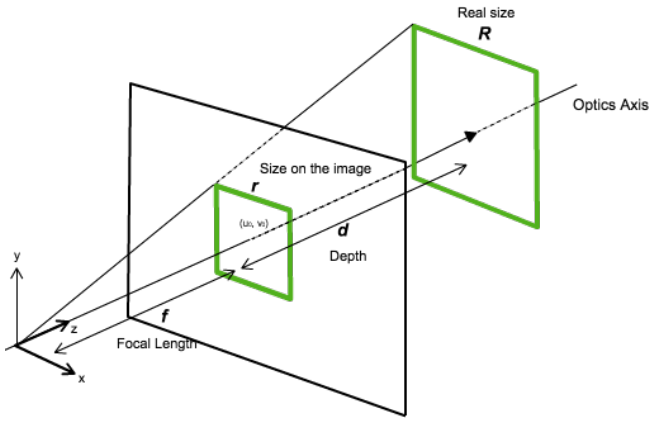
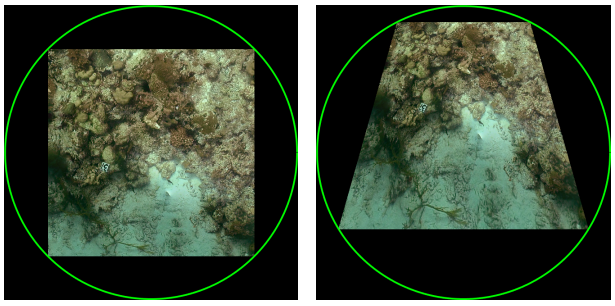


Fig. 4. Illustration for the Pinhole camera model

pixel $= 1\ mm$, so the converted size of the object on the frame plane can be acquired.

At the beginning, we simply used the upper line of the object for r and R. However, it has been shown that this is incorrect if the affine or even the perspective transformation is involved, which is very common in the real object recognition task. To solve the problem, we set r and R to the perimeters (sum of the length of each side) of the recognized object in the image plane and the real case separately.

As we know from the plane geometry, the cyclic quadrilaterals with the same radius have the same perimeter. Hence, no matter how your object is moving, the perimeter of the object in the frame is always the same as long as it is moving with a fixed centre point (u_0, v_0). Fig. 5 illustrates the theorem with an underwater image. As we can see, no matter how the image is deformed inside the circle, the perimeter is preserved.



(a) Orthogonal view (b) Perspective View

Fig. 5. Illustrations for the geometry fact that the perimeter is kept the same after applying perspective transformation

*4) **Marking objects on the map with Homography**:* As we mentioned earlier, when the object is detected the object recognition node publishes a custom message with a flag field set to *true*. In order to put a marker with the name of the detected object, we created a subscriber to the custom message in the SLAM node (Fig. 6 (Block-B)). Since the map visualization is independent from ROS, we cannot directly put markers on the map inside the SLAM Node. Therefore, we created a C++ class (we will refer to this class as *Recognition.class* further) that represents the recognized objects in ORB SLAM 2 package. This class is also included in the ROS Node. When SLAM ROS Node receives a message notifying that an object has been detected, we create an instance of the *Recognition.class* with the parameters that came with the message. In order to process these kind of instances we modified the source code of ORB SLAM 2 to process *Recognition.class* objects along with the RGB and depth images. Mostly, the modification took place in the tracking process, which we have discussed in Section V. We also had to add a functionality for Pangolin visualization module to plot the positions and names of the objects.

Until now, we only know the positions of the objects w.r.t. the camera. Before plotting the object on the map we have to find its position in the world frame. In order to do so, we obtained the pose (rotation + translation) of the camera when the object was being detected, which is described as the transformation matrix. Then, the position of the object is computed using the following equation:

$$\underbrace{\begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & t_x \\ r_{2,1} & r_{2,2} & r_{2,3} & t_y \\ r_{3,1} & r_{3,2} & r_{3,3} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{T_c} \times \underbrace{\begin{bmatrix} 0 \\ 0 \\ D_e \\ 1 \end{bmatrix}}_{p_{object}} = \underbrace{\begin{bmatrix} r_{1,3} \times D_e + t_x \\ r_{2,3} \times D_e + t_y \\ r_{3,3} \times D_e + t_z \\ 1 \end{bmatrix}}_{p_{world}}$$

where T_c - is the camera transformation matrix that shows how it is rotated and translated from the origin of the world frame; p_{object} and p_{world} - are object position in camera and world frames respectively; D_e - is the estimated depth. Then we update the corresponding object coordinates with the new computed world frame coordinates.

Now, by using the homography matrix, the 3D position of the object with respect to the world coordinate has been found. AS a result, we can directly put the marker with the name on that position in the map.

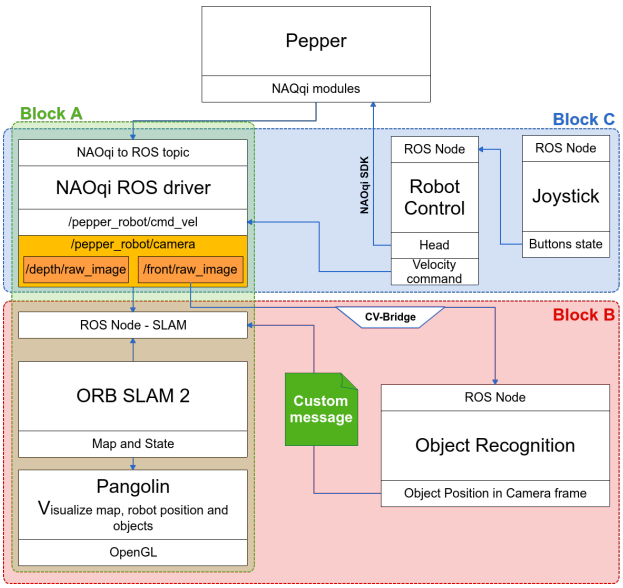


Fig. 6. System architecture.

C. Additional features

In this section we are going to talk about the additional features that are essential for performing SLAM and making the whole system faster and more practical.

1) ***Robot control with a joystick****:* The first thing that has to be mentioned is the robot control. This is the main module that is used for moving the robot in an indoor environment for building the map. This task is executed with the help of a joystick. The usage of a joystick ensures full control of the robot for SLAM. Additionally, by controlling the robot manually we can assure that all the necessary areas of the environment are covered and put in the map.

ROS provides a generic teleoperation tools [34], which is a simple library that reads commands from a joystick and publishes a vector with buttons state. In order to make it work with our robot we created a controller ROS node, that subscribes to the joystick node. Then, depending on the pressed button we define linear and angular velocities for the robot and send them to the /pepper_robot/cmd_vel topic. By sending velocity commands to that topic we can control the robot base. However it is worth mentioning this does not allow to control other joints of the robot.

For controlling the robot head we used NAOqi SDK directly inside our Robot Control node. First, we retrieve the current position of the head when a button, which was mapped to head movements, is pressed. Then, depending on the movement direction we calculate the final position of the head (in degrees). Next, using the *ALMotion* NAOqi module we send a command to the robot. Additionally, we programmed two more buttons that send the robot to *Rest* and *Active* status, which is implemented using NAOqi SDK as well.

The general overview of the robot controlling component of the system is illustrated in the Fig. 6 (Block-C).

The implementation details can be found in `/joy_pepper/scripts/joypepper.py`

2) ***Map saving & loading****:* Once the map of the environment has been built it is important to be able to reuse it. Saving the map becomes an important task due to the short working period of Pepper joints (they overheat after 2 hours). The implementation of the ORB SLAM 2 does not provide a functionality that allows to save the built map and load an existing map. In order to fill this gap and allow Pepper to continue the map building process, we have included this feature to our system.

First, a very naive method has been implemented where we save all the keypoints, keyframes and corresponding bag of words for each keyframe of the map into a text file. To reuse it we load this text file and parse it. This method appears to be very slow and inefficient, as expected, due to the large file size. Moreover, the processes of writing/reading from a text file are known to be slow.

Another way of solving this problem was saving all the instances of the C++ objects into a binary file, which is a well known strategy in programming called *serialization*. For ORB SLAM 2 there were already some research going on about this [35], where serialization and deserialization have been used for saving and loading the map. However, this has been implemented only for Monocular SLAM, and we implemented it in a similar manner for SLAM with RGB-D camera. More details can be found in the codes `Map.cc KeyFrame.cc MapPoint.cc` in our Github folder `/orb_slam2/src`

3) ***Fast vocabulary loading****:* For loop closing and camera relocalization the authors of ORB SLAM 2 used bag of words place recognition model [36]. This model uses a vocabulary of visual words, which have been built using a huge database of images. Every time when ORB SLAM 2 is launched it takes some time loading the vocabulary, because the vocabulary is saved as a text file which contains more than a million lines. This issue makes the start-up process very slow, and therefore the serialization for the vocabulary has been implemented in a similar manner as in the map serialization [35]. The source code can be found `/tools/bin_vocabulary.cc`

4) ***Object following / avoiding****:* We also implemented object following and avoiding functionality for Pepper. The main idea of this feature is to allow the robot to continuously track an object but also avoid it by keeping a certain distance when the object is too close.

The application works in the following manner: if the estimated depth distance from Pepper to the detected object is larger than $60\ cm$, then Pepper follows the object at a predefined constant speed. On the contrary, if the distance is calculated to be smaller than $20\ cm$ Pepper avoids it by going backwards. Moreover, we also want the detected object to be at the center of the frame. In order to do so, we computed the displacement of the central point of the object from the frame center. Then, depending on this displacement we send an angular velocity command to the robot to minimize this difference.

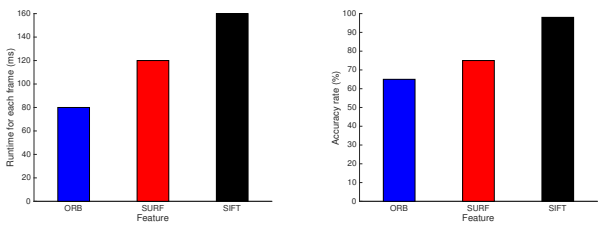
VII. RESULTS & DISCUSSIONS

In this section we discuss the object recognition, SLAM and the integration. We will discuss what we have accomplished as well as the comparisons with other object recognition and SLAM algorithms. For the real demonstration please check the link `https://youtu.be/evFsnWH_bpY`.

A. Object recognition

First of all, since we are using feature-based object recognition framework, the comparison of the recognition performance with various features should be discussed.

As we can see in Fig. 7, we choose and compare the performance of ORB, SURF and SIFT under our proposed recognition framework. In Fig. 7(a), SIFT takes longer time than the other two, but is under the acceptable updating time frame (200 ms). Nonetheless, when comparing the recognition accuracy (Fig. 7(a)), it can be clearly noted that the SIFT-based recognition achieves around 95%. Clearly, for our purposes, accuracy is more important than computational time. Hence, SIFT is chosen as our primal features extraction method for the object recognition task.



(a) Runtime (b) Accuracy rate

Fig. 7. Comparison of the recognition performance with ORB, SURF and SIFT. The implementation language is Python and 3 various objects (book, folder, T-shirt) are put in the database.

In Table 1, we compare our proposed SIFT + NN + RANSAC method with the well-known Haar Cascade method [9]. As we can notice, our method outperforms the Haar Cascades almost in all the cases.

Haar Cascades method requires a long time to train one object, while our method does not require training and needs only one image per object. This makes the system more flexible and easy to use by allowing users to modify the database just by adding/removing images of objects in one folder. By using the proposed method, no learning and training times would be required. The performance of our method also appears to be much more consistent than the other method and barely has false alarms.

Table 1: Comparison of Haar Cascade and our proposed method (SIFT + NN + RANSAC)

	Haar Cascades	Proposed
Training time	✗	✓
Detection consistence	✗	✓
False alarm	✗	✓
Rotation Invariant	✗	✓
Detect with partial info	✗	✓
A large number of objects	✓	✗

From the extensive experiments, when recognizing the same still object, we found out that the recognition accuracy of our method is almost 100%. In contrast, the accuracy of Haar Cascades is less than 60%, and false alarm and mis-detection may happen even in between two consecutive frames. Indeed, for Haar cascades the more negative / positive samples we use for training, the better recognition rate we obtain. However, the training time will also increase dramatically.

An important improvement of our method is enabling the object rotation-invariance. It turns out that the Haar-like feature does not evidently have the capacity of dealing with rotated object, unless a huge amount of samples with various angles have been used for training. Even though the training dataset is huge, we are not guaranteed with a decent result. SIFT instead is mainly famed for the rotation-invariance property. Fig. 8 undoubtedly shows that our method can cope with all kinds of rotational movements.

It is worth mentioning that our proposed method can still

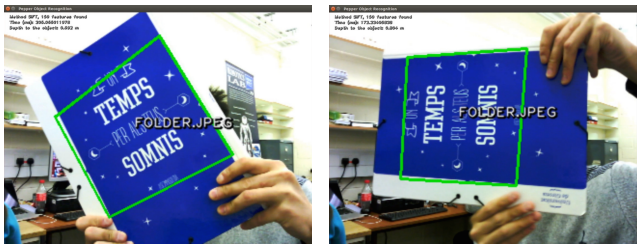


Fig. 8. Illustrations for the rotation invariant of our object recognition

recognize the objects properly with only partial details of an object. As we can notice from Fig 9, our method can still recognize correctly when around 30% of the folder has been covered. As we can remember in the Section IV, this improvement is due to the proper threshold that we set for the number of inliers. As long as the inliers number acquired from RANSAC is larger than 15, we say an object has been detected. For Haar Cascades, the object is not able to be recognized at all even if the covered portion is really small.

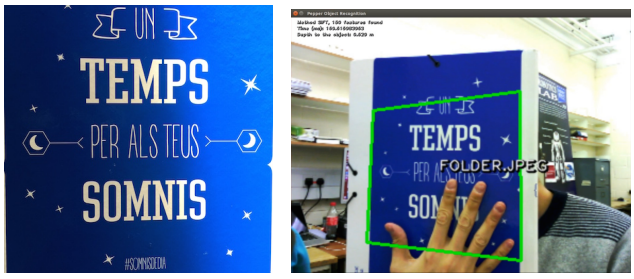


Fig. 9. Illustration that our object recognition can work with partial information

Finally, the main problem of our proposed method is that the recognition will be much slower when the number of the objects inside the database increases. For future work, to improve this part, the Bag-of-Words + SVM training scheme can be included. However, with regard to a home service robot which always stays indoors, it is already sufficient for Pepper to recognize a limited number of object.

B. SLAM with Object recognition

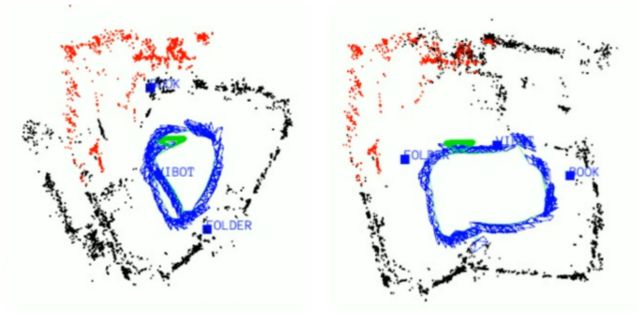
The experiments have showed that the ORB SLAM 2 outperforms its predecessor and LSD-SLAM in terms of initialization and depth estimation accuracy. LSD-SLAM provides dense reconstruction of the scene, however in an indoor environment it will cause lots of noise due to the inaccurate depth estimation of the points. Therefore, ORB SLAM and ORB SLAM 2 algorithms showed better performance in mapping an indoor environment. It should also be mentioned that LSD-SLAM is oriented for Large Scale environments, whereas ORB SLAM can be applied for both outdoor and indoor environments.

Table 2 summarizes the comparison of the SLAM algorithms that we tried for the implementation as well as the improved SLAM version that we are using to which we added extra features to the ORB SLAM 2 implementation.

Table 2: Comparison of SLAM algorithms				
	LSD	ORB	ORB-2	Ours
RGB-D support	✗	✗	✓	✓
Fast initialization	✗	✗	✓	✓
Accurate localization	✗	✓	✓	✓
Map saving & reusing	✗	✗	✗	✓
Fast vocabulary load	✗	✗	✗	✓
Recognition + SLAM	✗	✗	✗	✓

It can be clearly seen that the final result we obtained is the best among the others. As it was explained before, the most important features that include map saving and reusing play a significant role while performing SLAM with Pepper. Fast initialization for the tracking process is also achieved by leveraging depth camera as well as the decrease of the launching time due to the serialization of the vocabulary.

The results after running SLAM + object recognition are illustrated in Fig. 10. Here we can observe that the map on the left (Fig. 10(a)) is a preliminary result that has been obtained before the loop closure. When the robot arrived to its initial position, the system closed the loop and reconstructed a map of the environment as well as the trajectory of the robot as shown in Fig. 10(b). Blue markers on both images represent the inserted keyframes and the red (active) and black (inactive) points are the keypoints.



(a) Before (b) After

Fig. 10. Illustrations for the loop closing of ORB-SLAM2

After the loop closure we saved the map and reloaded it again to perform only localization of the robot and to test object recognition and localization on the map. The result of this test has been shown in the Fig. 11. From the top-left image we can observe the previously built map and the robot position as well as the position and label of the recognized object, which is shown in the top-right image. The bottom image shows the robot and the part of the environment where we performed our tests.

VIII. FINAL REMARKS & FUTURE WORK

In this section we summarize the results of our project and also introduce possible areas for the future work.

One of the main aspects is that an innovative application integrating a robust object recognition algorithm with a modified ORB SLAM 2 was proposed. This system was

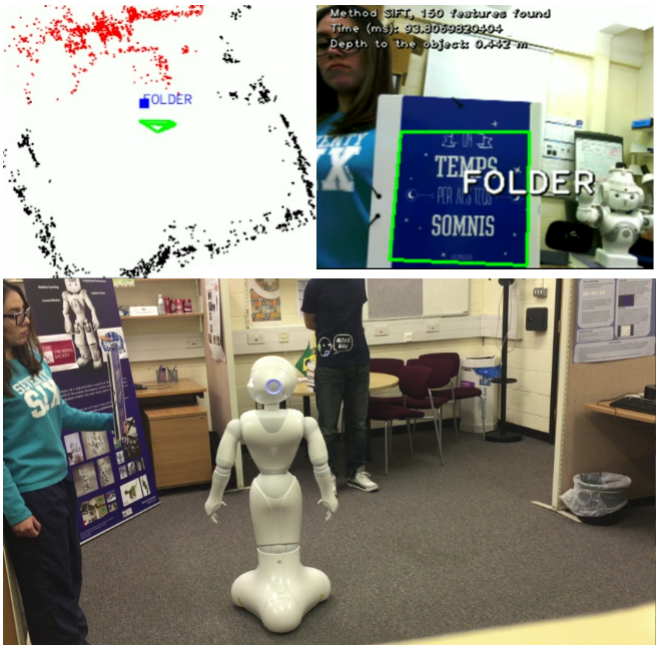


Fig. 11. Putting markers of detected object. Top-left: Map and the inserted marker; Top-right: Recognized object; Bottom: True location of the robot and object.

implemented and successfully tested on the humanoid Pepper robot under the scheme of European Robotics League.

As a summary, for the object recognition algorithm, SIFT features were extracted and then matched using kd-tree nearest neighbour search. Then, whether an object was recognized or not is decided through RANSAC. The algorithm has shown its robustness through its consistent detection, high accuracy without false alarm, and rotational invariance, etc.

Regarding the SLAM application, we have modified and improved the open source ORB SLAM 2 in following ways: enabling the map saving and reusing it, accelerating the vocabulary loading and most importantly, integrating the object recognition.

The whole system is successfully working on Pepper despite of the poor quality sensors, especially the cameras' resolution and frame rate as well as the joint overheating problem.

Finally, some future work can easily be implemented on top of our proposed application, for example:

- Autonomous control of the velocity while building the map.
- Add path planning algorithms (e.g. Rapidly-exploring Random Tree, Rotational Plane Sweep, etc.).
- Make Pepper to go to the marked position of a certain object and be able to grasp it and take the object back to the initial position.

ACKNOWLEDGEMENT

We would like to thank our project supervisor Dr. Mauro Dragone for his helpful guidance and support in the process, as well as Dr. Yvan Petillot for his comments and advice to our presentations.

We also thank Raul Mur-Artal for his outstanding ORB SLAM 2 algorithm, Bence Magyar for his advice of using Joystick teleop, and José María Sola Durán for his object recognition code framework.

REFERENCES

[1] ALdebaran Softbank Group. *Pepper Documentation NAOqi*. ALdebaran Sofbank Group.

[2] K-K Sung and Tomaso Poggio. Example-based learning for view-based human face detection. *IEEE Transactions on pattern analysis and machine intelligence*, 20(1):39–51, 1998.

[3] Michael J Swain and Dana H Ballard. Color indexing. *International journal of computer vision*, 7(1):11–32, 1991.

[4] Bernt Schiele and James L Crowley. Object recognition using multidimensional receptive field histograms. In *European Conference on Computer Vision*, pages 610–619. Springer, 1996.

[5] Jan J. Koenderink and Andrea J van Doorn. Generic neighborhood operators. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(6):597–605, 1992.

[6] Oskar Linde and Tony Lindeberg. Object recognition using composed receptive field histograms of higher dimensionality. In *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, volume 2, pages 1–6. IEEE, 2004.

[7] Henry Schneiderman and Takeo Kanade. A statistical method for 3d object detection applied to faces and cars. In *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*, volume 1, pages 746–751. IEEE, 2000.

[8] Serge Belongie, Jitendra Malik, and Jan Puzicha. Shape matching and object recognition using shape contexts. *IEEE transactions on pattern analysis and machine intelligence*, 24(4):509–522, 2002.

[9] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–511. IEEE, 2001.

[10] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. SURF: Speeded up robust features. In *In ECCV*, pages 404–417, 2006.

[11] Stefan Leutenegger, Margarita Chli, and Roland Y Siegwart. BRISK: Binary robust invariant scalable keypoints. In *2011 International conference on computer vision*, pages 2548–2555. IEEE, 2011.

[12] A comparison of keypoint descriptors in the context of pedestrian detection: FREAK vs. SURf vs. BRISK, author=Schaeffer, Cameron, journal=Cité en, pages=12, year=2013, publisher=Citeseer.

[13] Michael Montemerlo, Sebastian Thrun, Daphne Koller, Ben Wegbreit, et al. FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *Aaai/iaai*, pages 593–598, 2002.

[14] Danping Zou and Ping Tan. CoSLAM: Collaborative visual SLAM in dynamic environments. *IEEE transactions on pattern analysis and machine intelligence*, 35(2):354–366, 2013.

[15] Jakob Engel, Thomas Schöps, and Daniel Cremers. LSD-SLAM: Large-scale direct monocular SLAM. In *European Conference on Computer Vision*, pages 834–849. Springer, 2014.

[16] Raul Mur-Artal, JMM Montiel, and Juan D Tardós. ORB SLAM: a versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015.

[17] David G Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999.

[18] Jeffrey S Beis and David G Lowe. Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. In *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, pages 1000–1006. IEEE, 1997.

[19] S Yen, C Shih, H Chang, and T Li. Nearest neighbor searching in high dimensions using multiple KD-trees. In *Proceedings of the 10th WSEAS International Conference on Signal Processing, Computational Geometry and Artificial Vision, Stevens Point, USA*, pages 40–45, 2010.

[20] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[21] Raúl Mur-Artal and Juan D. Tardós. ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras. *arXiv preprint arXiv:1610.06475*, 2016.

[22] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. ORB: An efficient alternative to sift or surf. In *2011 International conference on computer vision*, pages 2564–2571. IEEE, 2011.

[23] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *European conference on computer vision*, pages 430–443. Springer, 2006.

[24] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: Binary robust independent elementary features. In *European conference on computer vision*, pages 778–792. Springer, 2010.

[25] Paul L Rosin. Measuring corner properties. *Computer Vision and Image Understanding*, 73(2):291–307, 1999.

[26] Chris Harris and Mike Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, page 50. Citeseer, 1988.

[27] Mark Everingham, Andrew Zisserman, Christopher KI Williams, and Luc Van Gool. The PASCAL visual object classes challenge 2006 (voc2006) results, 2006.

[28] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. Epnp: An accurate o (n) solution to the pnp problem. *International journal of computer vision*, 81(2):155–166, 2009.

[29] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. ROS: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[30] Natalia Lyubova. Pepper bringup plugin. `https://github.com/ros-naoqi/pepper_robot`, 2016.

[31] Steven Lovegrove. Pangolin. `https://github.com/stevenlovegrove/Pangolin`, 2016.

[32] OpenGL. *OpenGL official documentation*. OpenGL.

[33] James Bowman Patrick Mihelich. cv_bridge. `http://wiki.ros.org/cv_bridge`, 2016.

[34] Bence Magyar. A set of generic teleoperation tools for any robot. `https://github.com/ros-teleop/teleop_tools`, 2016.

[35] Raul Mur-Artal. ORB SLAM 2 implementation, modified by a github user @poine. `https://github.com/poine/ORB_SLAM2`, 2016.

[36] Dorian Gálvez-López and Juan D Tardos. Bags of binary words for fast place recognition in image sequences. *IEEE Transactions on Robotics*, 28(5):1188–1197, 2012.