Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione Corso di Laurea in Ingegneria Informatica

Tecniche per la rilevazione automatica marker-less di persone e marker-based di robot all'interno di reti di telecamere RGB-Depth

Laureando:
Nicola Ristè

Relatore:
Ch.mo Prof. Emanuele Menegatti

Correlatore:
Ing. Matteo Munaro

Anno accademico 2015/2016

Abstract

The problem of detecting known objects in an image or in a video stream is very important in computer vision. To achieve this task a machine must have a model of the object encoded within it. How this model is created, referenced and matched with a real world sample is what differentiate an object detection strategy from another.

Fiducial markers like April Tag are widely used in industrial contexts for their reliability and relative simplicity of implementation. Fiducials are, by design, always identical in every aspect (even size) to the model the machine has been given, only relative position and orientation to the camera change from one instance of the problem to the other. Also there can be differences in illumination and occasionally partial obstructions. Also, differently from retroreflective rigid-body markers, this kind of markers can't be continously detected if the target rotates on its vertical axis, unless there are enough strategically placed cameras around the target. So we had to design a composite marker such that even with a single camera a moving/rotating robot target can be tracked seamlessly and precisely. In our implementation multiple cameras detections are fused with a special Kalman Filter so that detection from multiple cameras of the same object lower uncertainty on the position.

On the other hand, more complex object detection relies on a model that must be, somehow, more flexible. In fact, unless we are trying to detect a quite homogeneous class of object (e.g. road signs), the system, to work correctly, must take in account, manage, exploit, or simply ignore any small discrepancy that can occur among real world objects that are part of the same class, but are not visually identical (e.g.: cars, people), furthermore, visual difference introduced by different points of view must be accounted for (perspective distortion).

In addition, if one wants to move a robot in a crowded evironment with many people randomly moving in it without putting both parts in harm's way, the system must know at every moment the absolute position of the robot in the evironment and also of the humans standing in it.

In this work we test the feasibility and the behaviour of a new type of camera for the OpenPTrack network, a mobile camera mounted on a people-following robot and dinamically registered in the camera network via the marker.

Contents

Introduction

The problem of detecting objects in an image or in a stream of images is a well known problem in the field of computer vision and one of the most studied. It consist of detecting a particular object in an environment taking as input a representation of the enviroment with as much information as possible (color information, depth information) and compare it with a model of the class of objects we are looking for. This must have enough information to be characteristic of the class, so to be sufficiently descriptive, but also not too much so keep the detection system computationally feasible. In literature there are many examples of people detection frameworks[4, 5], the main approaches comprehend ground based tecniques, that assumes people walking on a ground plane so any part of the image that is not "background" can be segmented and check if contains a person[1], or sliding window, in which a window at different sizes (in pixels) is slided among the whole image area, and every window is checked.

Both tecniques have limits, ground based tecniques, altough generally faster than s.w., can't detect people unless they are walking on the specified ground plane (e.g. people walking up stairs)[5]. On the other hand, sliding window tecniques, are much more computationally expensive as they have to compute confidence for every generated window[4].

In literature there are examples of hybrid tecniques, that uses geometric constraints to limit the sliding windows generation to the only windows that are compatible with some phisical properties of people walking on a specified ground plane[1] (i.e. min./max. height , people nor levitating nor compenetrating the ground) .

[1] A value of "confidence" is computed that is the probability that the particular window contains a person

Fiducial markers systems like April Tags, works in a two pass way, first the image is scanned for zones that may contain the figure of a marker in a low level fashion. Then, these zones are compared to an exact mathematical model of the marker of which we desire to know the precise position, after that, the exact pose is computed from the rectification of the quadrangle (homography). In literature we have many examples of tag detection frameworks like ARuCo or ARToolKit, we choose AprilTags for the better accuracy in environments with ununiform lighting and visual obstructions, the library also offer better performace in comparison with the ArUco library.

There are some examples in the web of composite fiducial marker detection and tracking and also the ArToolKit library provides templates for setting up a multimarker[12].

Once the position of all the agents involved is resolved with a certain amount of confidence our robot agent can be guided by a simple algorithm to follow people according to its programming with the assurance of a precise people tracking and self-locating.

With this work we have been able to track people and markers in an heterogeneous network of sensors and computers. Our system can track people in an environment made of machines with an arbitrary number and different types of cameras (i.e. RGB/RGB-D). The utilization of non depth-sensitive cameras requires another algorithm to be used, that exploits the parallel computation capability of nVidia Graphic Processing Unit instead of the standard CPUs. It is possible to obtain a decent detection speed on a cuda-Enabled device with a standard webcam.

We also experimented with an OpenPTrack detection node running on a mobile robot platform programmed to follow people, the camera is dinamically registered in the openPTrack camera network through the high mounted composite marker, and people detections are referred to the world frame like those from the fixed cameras.

A Software

ROS, OpenCV, PCL

Robot Operating System (ROS) is a collection of software frameworks for robot software development. It provides operating system-like functionality on heterogeneous computer networks. It includes libraries, developing tools and conventions to simplify robot code development. Through ROS we were able to get images and point clouds from the sensors, manage the work on different nodes and compute the pose of an object with ready-to-use libraries. Also, with its modular structure, multitasking can be achieved with little effort. In particular we used the OpenCV, Point Cloud Library, and TF modules.

OpenCV was created as library of functions and classes to represent and manipulate images in an efficient fashion. Since its release, many modules have been developed and optimized, and new functionalities added. Like for example modules for machine learning and gpu-enabled variants of the most resource demanding algorithms. Many operations like transformations, color conversion and a wide range of specific segmentation algorithms can be efficiently executed with specific API calls in a clean and consistent manner.

The Point Cloud Library is another library of the ROS suite. It can be considered an extension of OpenCV. It is used to process RGB-D data, in the form of point clouds, i.e. sets of points in a 3d coordinate system. It contains algorithm for 3d points processing, analysis, and many geometric transformations functions.

groundHOG

groundHOG is a software, published as external library, which implements a people detection algorithm developed in Cuda (cudaHOG). This library allows the developer to enable geometrical constraints that limit the search area for people detection within the image frame. This considerably speeds up the computation of the HOG confidences as we rule out many windows that we assume do not contain humans (e.g. People compenetrating the ground, walking mid-air, or that do not fall in the set up range of heights reasonable for a human).

Besides the geometric optimization, the parallel computation of the confidence for a single window gives notable speedup compared to other CPU implementations of sliding window tecniques.

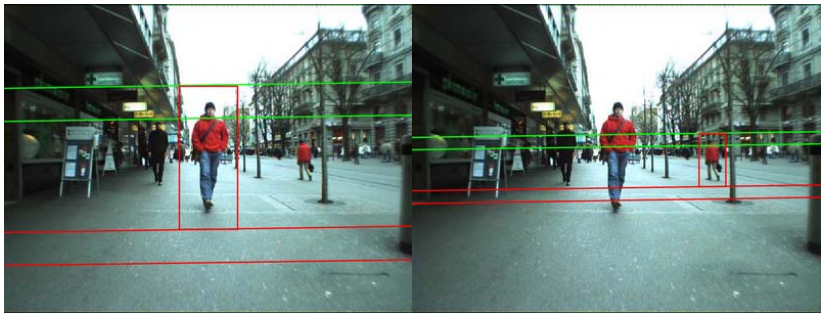

Figure 1: *The groundHOG corridor. For any scaling factor, the cudaHOG algorithm computes the confidence value only in the windows that respects the imposed geometrical constraints (i.e. Ground Plane Equation, min./max. height, etc.) through groundHOG's software interface.*

TF and Eigen

Tf is a package that lets the user keep track of multiple coordinate frames over time. A robotic system has typically many 3D coordinate frames that change over time. For example, a *world* frame that is integral with the ground, a *base_link* or *chassis* that is fixed with the robot body, and, if the robot has an arm with a gripper or other kinds of tools, there can be a frame called *end_effector* representing it. Tf keeps track of all these frames over time and provides APIs to query the relative position of every frame with any other in the tree, composing the transforms accordingly, and also within a limited time interval back in the past. Eigen is a template library for linear algebra that is used for geometrical operations on geometrical transformations and vectors.

Cuda

Cuda (Compute Unified Device Architecture) is nVidia parallel computing platform and Application Program Interface model that enables significant improvements in computing performance exploiting modern GPUs computing capabilities. It allows software developers to use a CUDA-enabled graphics processing unit (GPU) for general purpose processing, an approach known as GPGPU. The CUDA platform is a software layer that gives direct access to the GPU's virtual instruction set and parallel computational elements.

The CUDA platform is designed to work with programming languages such as C and C++. This accessibility makes it easier for specialists in parallel programming to utilize GPU resources, as opposed to previous API solutions like Direct3D and OpenGL, which required advanced skills in graphics programming.

With Cuda, the latest nVidia GPUs become an open platform like the CPU. Differently from CPUs , GPUs have a parallel architecture with many processors (Stream Multi-Processor) each composed of 32 Cuda Cores. The number of cuda cores in a graphic card specify how many parallel instructions can be executed concurrently. Also, the programmer must understand and make good use of the memory hierarchy (from Shared Memory (on-chip, the fastest but the smallest, accessible by all threads in a single block of threads) to Global Memory (Video RAM, the slowest unless coalesced access, but the largest). If an application is compatible with a parallel architecture, a GPGPU implementation can give a substantial speedup.

The Cuda library in addition to low level primitives for memory management and kernel launches, also provides higher level interfaces for some common programming tasks like for example sorting operations, transformations, reductions and many other linear algebra algorithms (Thrust library).

OpenPTrack

OpenPTrack is an open-source project launched in 2013 to create a scalable, multi-camera solution for people tracking, to support education, arts and cultural applications. It enables many people to be tracked over large areas in real time. With the advent of commercially available consumer depth sensors, and continued efforts in computer vision research to improve multi-modal image and point cloud processing, robust person tracking with the stability and responsiveness necessary to drive interactive applications is now possible at low cost. Based on the widely used, Robot Operating System (ROS), OpenPTrack provides:

- User friendly camera network calibration
- Person detection from RGB, infrared and depth images
- Efficient multi-person tracking.

For this thesis the modules for detection, tracking and camera calibration have been used. And we present the work that has been taken forward on the detection and the calibration module.

April Tags

Figure 2: April Tag, esempio

AprilTags is an open-source fiducial markers system used for various tasks (i.e. augmented reality, robotics and camera systems intrinsic and estrinsic calibration). Based on a lexicographic coding system, it is also robust to lighting, detection angle, and partial occlusions of the tag. The provided C++ library allows to compute position, orientation and ID of multiple tags w.r.t. the calibrated camera that is framing the scene. The detector was designed to run on standard VGA images (640 by 480 pixels). However, this resolution is too low to work with tags that are too distant from the camera. We optimized the code of the library to run faster using parallel computation and optimized functions as we decided to work on FullHD images that are much harder to compute.

B Hardware

Development and testing machine

Most part of the development was done on a Dell XPS PC with an Intel i7-2670QM, 8 GigaBytes of RAM and a GeForce 540M with 96 CUDA cores.

The i7 family's processors have a technology called "Turboboost". The processor base clock is 2.1 GHz, with turboboost enabled, the processor can overclock up to 3.1GHz if only one core is running max load. As the number of processesor running simultaneously goes up (loading average), the max possible overclocking frequency will decrease. Down to the base clock when the usage goes 100% for all eight cores.

Structured Light Sensors: Kinect mod.1 and 2

Figure 3: Sensori a luce strutturata di prima generazione: Asus xTion, Microsoft Kinect, Primesense Carmine

Figure 4: Kinect One Sensor

Calibrated stereo camera systems can compute the distance of a particular pixel in the frame of one of the cameras by resolving its correspondence in the other image. The algorithm scans the row with the same vertical coordinates in the other picture until it finds a similar pixel; the horizontal difference is called disparity. Disparity is inversely proportional to depth and can be used to compute it.

However this method has issues: a block matching algorithm is required to resolve the stereo correspondence, although this can be done efficiently with the latest hardware/software, it's still not good enough when working with poorly textured scenes (white walls), and the precision at a certain distance varies with the baseline.

On the other hand, a structured light scanner is a 3D scanning device for measuring the tridimensional shape of an object using projected light patterns and a camera system. This kind of sensors have rapidly gain the

attention of the computer vision expert community because they are cheap, reliable and (almost) ready to use devices for 3d perception.

Structured light sensors eliminate the correspondence problem by projecting structured light on the scene, a CCD camera then observes the deformations in the pattern caused by the shape of the objects and, by triangulation, calculates the depth of a particular pixel. 9x9 pixels codeword sub-patterns are used to resolve the correspondence between the whole projected pattern and the image points.

Unlike stereo camera pairs, they require a simpler calibration process but they have less operative range, because the projector pattern can be correctly detected only within a certain distance, and they don't work in open sunlight.

Initially we used first generation devices such as Microsoft Kinect v.1, Asus Xtion and Primesense Carmine, that provide 640x480 images at 30 frames per second, the main difference between these sensors is the different type of precision. While for example, the Kinect is quite balanced, the Carmine sensor in comparison provides better accuracy for measurements closer to the camera. Later, as we needed better resolution especially for the rgb images, we switched to the successive generation of Microsoft's sensor, the Kinect One. It provides FullHD 1920x1080 resolution images at 30 fps. However, to generate depth images in real time it requires a recent graphic card.

1 Definition of the problems

1.1 Multicamera Calibration

A multi camera network calibration process is meant to find the relative position (rotation and translation), of every camera w.r.t. the others and to a common reference frame usually placed on the ground called *world*. The origin of this reference frame will represent the (0,0) in the bidimentional grid where people is moving. This is usually done when working with stereo pairs, and we wish to set them to know the 3d position of a point in the field of view of the two sensors (stereo matching). The calibration process needs a checkerboard of known physical properties such as square dimension and disposition. Also, time synchronization between all the computers in the network is required for maximum precision.

When two cameras see the checkerboard the transformation between the two is estimated. Every sensor is then extrinsically calibrated with respect to another sensor, composing a tree of transformations which describe the whole network. Once all sensors have been added to the network and the checkerboard is placed accordingly to the desired world frame position/orientation, it's possible to save the calibration data.

Because the process is not error-free, OpenPTrack provides other tools to refine the calibration, based on more precise solvers for the chessboard corners alignment, or single person track matching between cameras that minimizes the offset between the tracks.

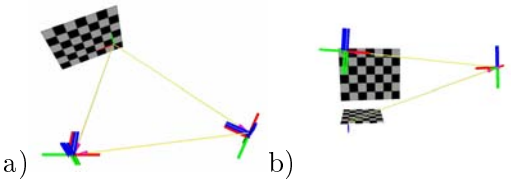

Figure 5: a) Pairwise calibration b) World frame calibration

1.2 Ground Plane Equation Extraction

In openPTrack's people detection module, the ground plane equation is of pivotal importance. The assumption of people standing and moving on a ground plane, allows the people detection routine to correctly segment it and cluster the points belonging to humans. Substantially speeding up computation.

Planar segmentation of Point Clouds is a known problem for which a number of approches exists in literature. Many of these are based on RANSAC (RANdom SAmple Consensus) model. Ransac is an iterative method to estimate parameters of a mathematical model (in our case, the geometric equation of a plane) from a set of observed data (point cloud) that contains outliers (all non ground plane points). These methods, correctly segment planar components, but since they are designed for unorganized Point Clouds, they are much slower than methods that exploits point cloud organization. In fact, real time performance is achievable on some systems.

Organized Multiplane Segmentation is a method proposed in [1] to efficiently segment organized point clouds. Exploiting organization, many time consuming operations like nearest neighbour search become much faster. In this strategy every point is sequentially processed and the plane-model fitting is deferred tilL the end of the segmentation process. Segments are generated through a two pass region growing labelling process. First, labels are assigned to the points with respect to the surface normal, then the labels are joined with union find so that every region gets the lowest applicable label. Once segmentation is done, plane-fitting is launched and for every planar surface the best fitting plane equation is computed via RANSAC.

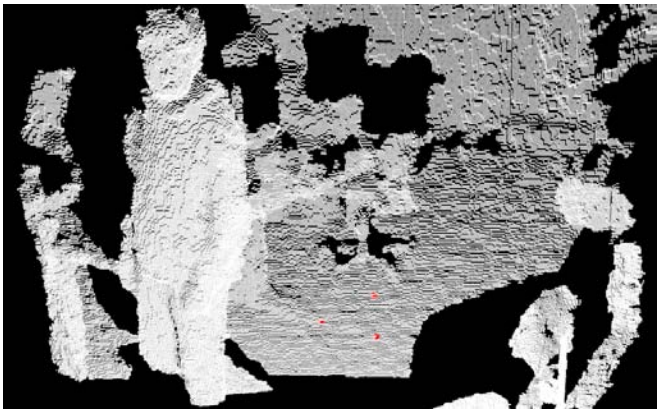

Figure 6: Manual Segmentation, three unaligned ground point must be clicked to segment the ground

1.3 Visual Marker Detection

Robot self-localization, that is to determine its position in a known or unknown environment, is a fundamental task for a mobile robot. Literature is full of examples of autonomous localization achieved through sensors directly mounted on the robotic unit (SLAM, Visual Odometry) that are currently state of the art. Our idea is to delocate (self) localization from the robot and use the computational power of the other machines in the network to achieve this task in a distributed fashion. On the other hand, we also need to keep the computational load for this particular task reasonably low as they will likely to be occupied with others (People Detection / Tracking).

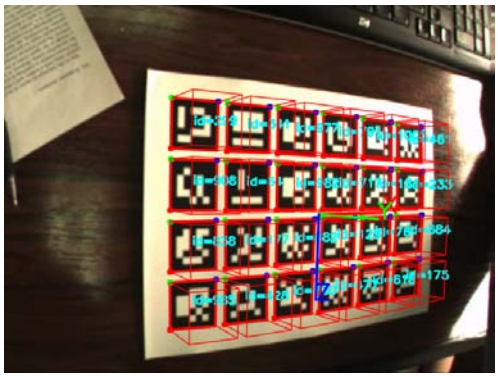

Figure 7: ArUco library

Visual fiducial markers are objects with known phisical dimensions that are placed in the field of view of an imaging instrument to be used as reference for measurements. Retroreflective markers are widely used in moving picture industry to realize the so called "performance capture" on actors, fiducial markers are gaining popularity among the augmented reality community for the good compromise between simplicity, precision and accuracy. Visibility is a downsize of fiducials, in fact, the likelihood of a detection at increasing distance depends, without taking in account illumination or partial obstructions, on the perspective distortion due to relative orientation to the camera. Instead of using a 3d retroreflective marker, for which a special setup of cameras with an infrared light projector and an infrared sensitive

camera would be required, we designed a cube shaped marker with face sized fiducial markers on every side, a sensor fusion algorithm can return the absolute pose of the whole cubic marker given any face/marker that one or more camera or is able to see.

1.4 People Detection and Tracking for mobile robots

To detect and track people (or known objects) in video streams is a key technology in many fields of robotics like, road safety, video-surveillance, human-machine interaction or image and video indexing on the web. It is also one of the hardest problems in computer vision and a real scientific challenge for realistic and complex scenes.

We inspect the people tracking problem from the perspective of an autonomous robot acting in populated environments. Such a robot must be able to dynamically perceive the world, distinguish people from other objects in the environment, predict their future positions and plan its motion in a human-aware fashion, according to its tasks. The tracker proposed by Munaro[5] utilizes a three-terms joint likelihoods to limit drifts and ID switches, and an online learned appearance classifier that robustly specializes on a track while using other detections as negative examples. The HOG confidence from the detector is used to robustly initialize new tracks when no association with existing tracks is found. The tracker uses input detections from one or more detection modules and solves the data association problem as the maximization of a joint likelihood encoding the probability of motion (in ground plane coordinates) and color appearance, togheter with that of being a person. An Unscented Kalman Filter is exploited to predict people position and velocities along the two ground plane axes.

1.4.1 People Detection: Sliding Window vs Ground Based

The main approaches to people detection examined in this work are sliding window- and ground-based techniques.

The first method run as follows: the image to be searched for people is divided in a number of windows of a certain initial scale so that every windows covers a precise area of the frame, the process is repeated for a certain number of larger scales of the windows. The horizontal/vertical offset applied at a single scale to move the window and the number of scales to be searched are the parameters that affect the time for detection. Some implementations of this technique exploits some geometric properties of the real world to reduce the number of windows to be generated, given the the ground plane equation these algorithms are able to avoid searching in those windows that are incompatible with the phisical/geometric constraints of the real world (ideally, people not touching the ground or that are compenetrating it). The algorithm then assign to every window a "confidence" that is a metric of how much the particular window is likey to contain a human.

If some windows that are close one by another reach the threshold of confindence needed to be labeled as "containing a human", than this detections are fused in one by an algorithm of "non maxima suppression". This returns the window among the cluster that have the maximum confidence.

Ground Based detection uses another approach: given a point cloud of the scene and the ground plane equation, first the points of the ground plane are segmented and removed together with walls so that we are left with a set of cluster that may or may not be/contain people. Moreover, some of the clusters produced by the same person may not be connected. Once geometrically valid clusters are extracted, a HOG based detector is launched on the area of image containing the 3d bounding box.

Ground based techniques are quite faster than sliding windows, as generally, applying or exploiting geometric constraints substantially speeds-up detection, as big portions of the image are simply skipped by the detector funtion. This can be a key feature in some applications but also a limitation for others as we will see further in the dissertation.

1.4.2 PCL::People::GroundBasedPeopleDetector

The Point Cloud Library provides a module for robust and real time people detection from RGB-D images. This tecnique rely upon a method of subclustering specifically designed to detect people in compact groups or near the background.

The process is divided in four phases:

- Voxel grid filtering: It consists of a smart subsampling of the point cloud. At every frame, the space is subdivided in a certain number of voxels (volumetric picture element) depending on the resolution set up and all the points in every voxel are approximated to their centroid. The default value of voxel size for this phase is 0.06 meters. This, in addition to shrink the size of the point cloud to an order of magnitude, also gives us a constant density point cloud (not dependant on the distance of sensor).

- 3D hierarchical Euclidean Distance based Clustering: The algorithm is based on the assumption that people are walking on a ground plane, so points belonging to it can be removed. The segmentation of the ground plane is done through a RANSAC based method that will be discussed in detail further on the dissertation.

- People Detection: Once we obtained valid clusters and extended the "theoretical bounding box" that contains the cluster to the ground, a HOG based detector is run on the portion of picture correspondent to the teoretical bounding box.

There is also an optional phase in which the point cloud is rotated so that the ground plane is parallel with the optical axis of the camera, that will be discussed later.

Figure 8: Left: Before NMS. Right: After NMS

1.4.3 CudaHOG

Figure 9: HOG+SVM detection pipeline

cudaHOG is a CUDA C implementation of the people detection pipeline described in [4]. It uses cuda API's calls to parallelize the evaluation of the support vector machine for a certain window. The sliding window algorithm produces many series of windows, each series at a different scale, SVM evaluation is performed on these portions of images called Regions Of Interest (ROI). ROIs are generated "sliding" a fixed aspect ratio window along the width and heigth of the picture and at different scales. Additional windows are generated so that a portion of these can lay out of the image (padding) so that also objects partially out of frame can be detected. In this application, the HOG confidence is a function, defined between all the pairs of points $P1$, $P2$, such that $P1.x < P2.x$ and $P1.y < P2.y$. Confidence is a metric that states the probability that the window with $P1$ as upper left corner and $P2$ as the lower-right corner contains a person. In some cases more than a window may be generated on a single person, a non-maxima-suppression algorithm is run so that these detections are clustered into the one with the higher value of confidence.

It's immediately clear that in this manner a very large number of samples is generated, that is why although CPU implementations of this algorithm are state-of-the-art for quality of detection and accuracy in many applications, they are not feasible for real-time or time sensitive applications in general. Parallel implementation with Cuda gives a sensible speedup to the algorithm. The cudaHOG library can be initialized with two parameters, hog_start_scale and hog_scale_step. These tell the window generation algorithm how many ROIs must be generated in this way: the start scale is the first scale that is applied to the svm model (with a 64x128 pixel window a human can be detected if its "footprint" on the image is delimited almost perfectly by the window. If the footprint is slightly smaller , then the chances

to detect it are smaller, the same happens if it is slightly larger, but in this case a bigger window at the successive iteration would detect it). The hog scale step defines the increment to be applied at the scale factor at the next iteration up to a default end value.

1.5 The Pioneer P3-AT people following module

The Pioneer robot has been programmed with an additional ROS module, written in Python, that implements a simple people following routine. I.e. to keep the first person detected at the center of the frame and keep the robot within a certain distance from the person followed. The max moving speed of the robot is 0.7 m/s. An URDF (Unified Robot Description Format) that describes the robot frames is included and an odometry software can approximately compute the final position of the robot after a moving order is issued. The odometry is modeled like a skid steered platform.

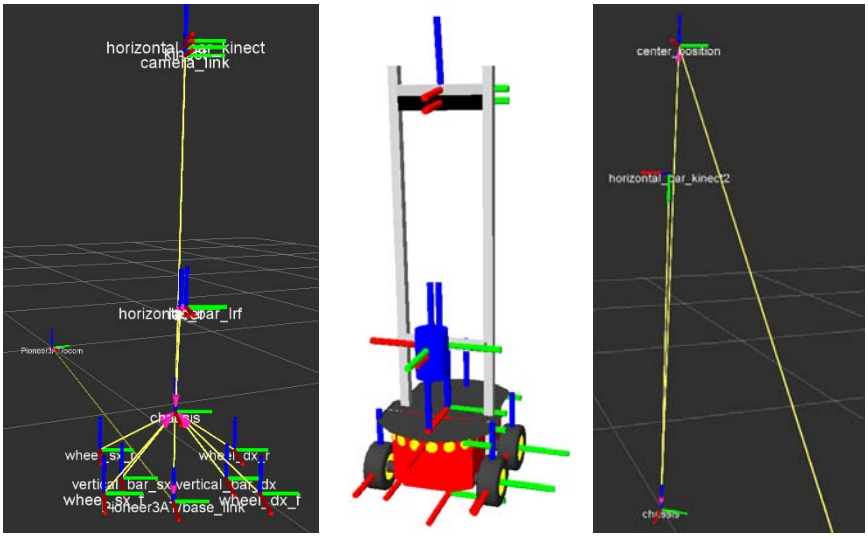

Figure 10: Robot frames for the pioneer robot visualized in rviz: From left to right a) Tf as described in the URDF file for the Pioneer robot, with the frame relative to the original position (from wheel odometry). b) URDF Model with tf frames (without the cubic marker and the Kinect One bar). c) New tf frames that connect the cubic marker frame and the world frame (not in picture) with the chassis and the Kinect One bar.

2 People Following in heterogeneous camera network

2.1 Multicamera Setup

We present the results we obtained while testing the system in a controlled environment. We set up the camera system as described above and recorded three datasets from the three synchronized machines.[2].

Figure 11: The IASLab setup, the camera in the foreground at the left is connected to the master node while the kinect on the right is linked to the "client" node

[2]The ROS tool *rosbag* allows to register topic's messages during executionsaving them in large *.bag* files. Afterwards, these messages can be re-played in a way that is totally transparent to the ROS environment. Because these messages have absolute timestamps, if the computers are well synchronized, playing back multiple bags simultaneously, will publish all messages from the different bags in absolute temporal order with no distincion of which bag file the mesage is from.

2.1.1 Camera Network

To test our implementation we set up camera network with two Kinect One sensors, plugged to two notebooks via a Gigabit ethernet cable. In our setup the "Master" node is an Lenovo Thinkpad (Intel i5, left side of the image) and the secondary node called "Polluce" is a Lenovo Y50 (i7, out of frame to the right). For these two cameras, in addition to an optimized camera calibration algorithm, we also appllied the calibration refinement routine, and used openptrack built-in background segmentation app when no people was in the scene to remove it and speedup detection.

The two nodes clocks, for tracking to run consistently, must be synchronized up to a tolerance of 33ms, that for a 30fps video is equal to the inter-frame period. We first tried synchronizing the machines to a public time server (*time.nist.gov* of the National Institute of Standards and Technology, U.S.A.) but, as we set up a local network with the computers and disconnected them from the internet, the sinchronization was lost over night. Therefore, synchronization over the internet with a central server is possible, although not reccomended. It's better to set one of the machines (i.e. the master) as ntp server and have the other computers use it as a reference for time. The procedure is thoroughly described in the OpenPTrack's user guide.

2.1.2 Mobile Camera

OpenPTrack is able to refer detection from fixed cameras to a common coordinate frame because the system knows the relative position of any camera with another from calibration. A mobile camera must be treated differently, as its position relative to the world frame changes over time. In this case the robot was programmed with a people following routine to track and follow the first person that got in the field of view of the kinect mounted below the cube. To insert the camera in the tree of transformations relative to the network we had to link it to the cubic marker. A direct link would be possible, however, we decided to link first the marker with the pre-existing

chassis frame of the robot and then link the latter with the newly mounted Kinect One frame. In this manner, we also linked the rest of the robot's frames described by the URDF model.

2.1.3 Tag Disposition

Our composite marker tracking system is designed to detect and report correctly, robustly and continously the position of the whole marker whichever face/tag the camera (or cameras) sees. Therefore we estabilished the origin of the cubic marker at the center of it, with the x-axis pointing toward the frontal face, y-axis pointing at the left face and the z-axis pointing upward, in a canonical robot-frame convention.

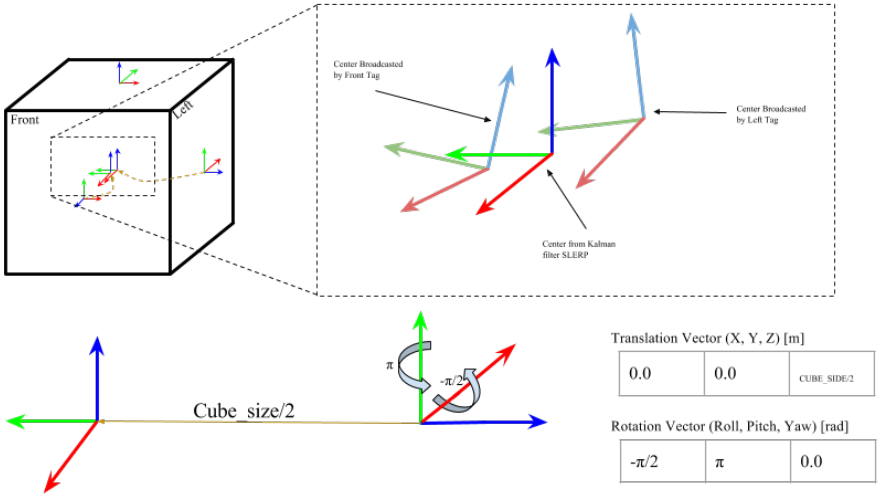

Figure 12: Centers intepolation and example of center broadcasting from the left face

The transform to apply to every face to get the center transform depends on the cube face and can be set from the center_transform launche file. If more than one face is visible (by the same or by another camera in the network) both center transforms are available, and the kalman filter for the

centers can fuse all the center detection into one. Altough apriltags detection system is quite accurate with the marker position there can be a little jitter on the orientation measurements. The kalman filtering adresses this problem. However, because the cube is made of cardboard and the tags are printed on stardard paper glued to the cube sides, their positioning, although sufficiently accurate all thing considered, is not perfect and the projected centers may not fall all in the exact same place and with the exact same orientation.

The error is anyway not large enough to affect substantially the center tracking. Worst case scenario, if one face is perfectly visible by one camera and the other has a more discountinued detection, the center can "jump" from the center projected by the better seen face to the Kalman-filter-generated virtual center. This can cause a little jitter in the center tracking. During our tests this situation occurred very sporadically and never in such a degree to affect the robot tracking.

2.1.4 Multicamera Calibration

A muti-camera network calibration is meant to find the relative position (rotation and translation), of every camera w.r.t. the others and to a common reference frame called "world". This is usually done when working with stereo pairs, and we wish to know the 3d position of a point in the field of view of the two sensors (stereo matching). The calibration process needs a checkerboard of known physical properties such as square dimension and disposition. Also, time synchronization between all the computers in the network is required for maximum precision.

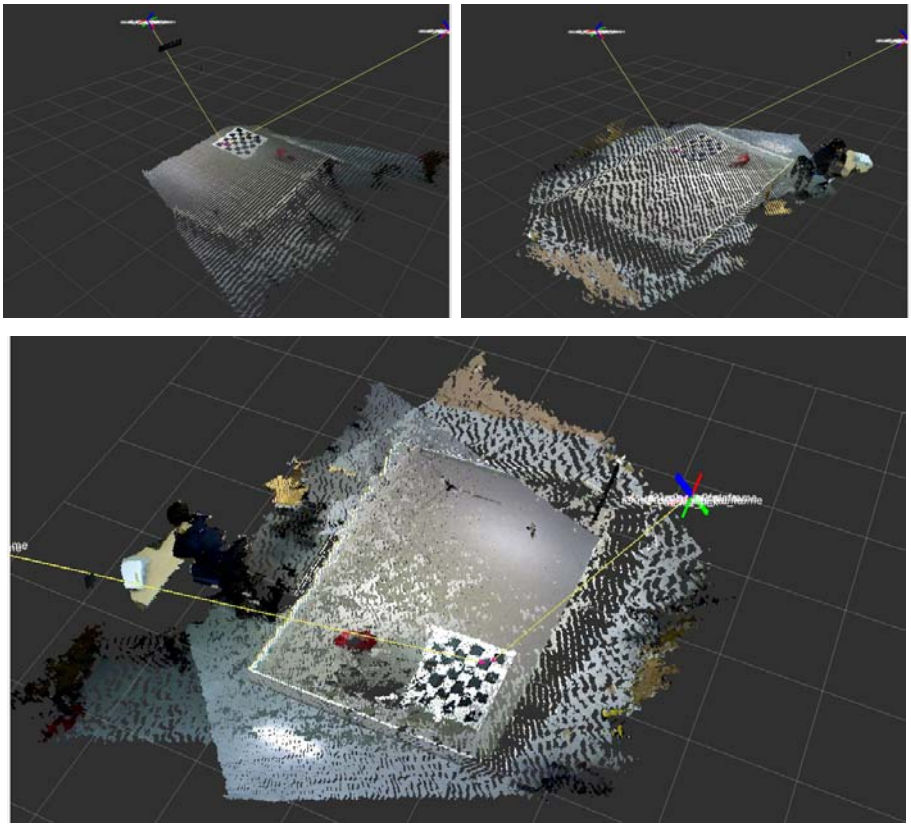

Figure 13: Example of Extrinsic calibration results, the point clouds from the two sensors are correctly overlapping

When two cameras see the checkerboard the transformation between the two is estimated. Every sensor is then extrinsically calibrated with respect to another sensor, composing a tree of transformations which describe the whole network. Once all sensors have been added to the network and the checkerboard is placed accordingly to the desired world frame position/orientation, it's possible to save the calibration data.

Because the process is not error-free, OpenPTrack provides other tools to refine the calibration, based on more precise solvers for the chessboard corners alignment, or single person track matching between cameras that minimizes the offset between the tracks.

2.1.5 Cloud Rotation

At the beginning of the work we discovered a problem that under certain conditions affected the quality of the detection. The detector was unable to detect people when the camera was high-mounted and with an high tilt angle. The detector has some parameters that determine the minimum and maximum heigth of an admissible 3d bounding box. The perspective distortion, introduced by camera tilt, inevitably alters the shape of the bounding box returned by the euclidean clustering routine. As a consequence of such bad proportions, the detected cluster height couldn't lay in the specified interval of validity and the BB was ruled out. It would have been possible to adjust the range of validity to include also these extreme cases, but we decided not to take this road to not affect the performance of the detector in standard conditions.

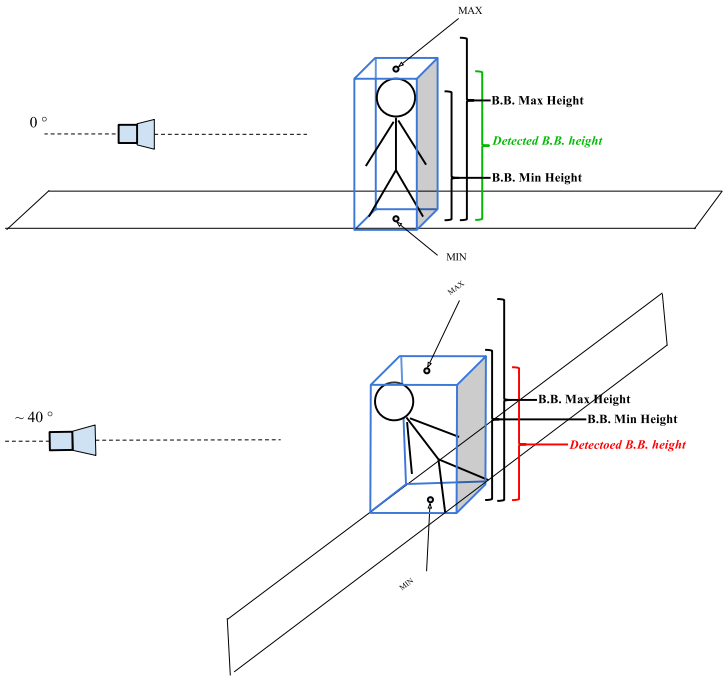

Figure 14: Point Cloud Rotation, the rotated bounding box is lower than the minimun height and is rejected

The solution we applied was to rotate the point cloud before clustering with the PCL built-in function *transformpointCloud(cloud_in, cloud_out, trasform)*, that applies an affine transform to the input point cloud. The rotation matrix is obtained via dot product between the ground plane normal and the camera XZ plane normal giving us the dihedral angle between the intersecting planes. The cross product between the two planes gives the rotation axis. The Eigen function, AngleAxis(dihedral_angle, rotation_axis) returns the transform to be applied.

$$\theta = \arccos \frac{P_g \cdot P_c}{|P_g| \, / \, |P_p|}$$

The dihedral angle between the x0z camera plane P_c and the ground plane P_g is computed with the formula above. And the final transform is obtained with the built-in function of the Eigen library $AngleAxisf(\theta, A)$. Also, the inverse transformation is computed and saved for later use. After detection is done, all we need to do is apply the anti-transformation to the bounding boxes characteristic points, so that we have detections in the un-rotated point cloud reference system.

2.1.6 Robot Tracking

Figure 15: The pioneer robot

Once the cube and the kinect2 are mounted on the robot, we proceeded to measure the relative position of the two w.r.t. the chassis of the robot. Then, we can broadcast these transformation with ROS. Finally, with all the coordinate frames linked toghether, openPTrack can refer the mounted kinect detections to the "world" coordiante frame like any other fixed-mounted camera.

2.2 Organized Multiplane Segmentation

2.2.1 Ground Plane Equation Extractor

In openPTrack's people detection module, the ground plane equation is of pivotal importance. The assumption of people standing and moving on a ground plane, allows the people detection module to segment and remove it to aid clustering the points belonging to humans and speed up computation.

Planar segmentation of Point Clouds is a known problemt for which a number of approches exists in literature. Many of these are based on RANSAC (RANdom SAmple Consensus) model. Ransac an iterative method to estimate parameters of a mathematical model (in our case, the geometric equation of a plane) from a set of observed data (point cloud) that contains outliers (all non ground plane points). These methods, correctly segment planar components, but since they are designed for unorganized Point Clouds, they are much slower than methods that exploits point cloud organization. In fact, real time performance is achievable on some systems.

Organized Multiplane Segmentation is a method proposed in [2] to efficiently segment organized point clouds[3]. Exploiting organization, many time consuming operations like nearest neighbour search become much faster. With this strategy every point is sequentially processed and the model-plane fitting is deferred till the end of the segmentation process. Segments are generated through a two pass region growing labelling process. First, labels are assigned to the points with respect to the surface normal, then the labels are joined with union find so that every region gets the lowest applicable label. Once segmentation is done, plane-fitting is launched and for every planar surface the best fitting plane equation is computed via RANSAC. Returning a list of planes equation in the Hesse normal form for plane equation.

[3]Organized Point Clouds that mimic the structure of a bidimentional matrix (image). They are such that every 3D point has a single corresponding 2d point in the image

2.2.2 Plane selection

With the technique described above, it was possible to realize a functional ground plane detector with a simple selection of the plane based on geometric properties of the segments (planes) found. We use the plane position and its relative orientation.

There are four detection modes:

- MANUAL MODE: The user is asked to select 3 or more points (strictly not aligned) from a PCL Viewer window and plane fitting is run on the last three points selected.
- SEMI-AUTOMATIC: The planar regions are segmented with OMPS and the valid ground plane candidates are highlighted based on inclination (that must be not vertical so that we can rule out walls). The user is asked to select the one to use as ground plane.
- AUTOMATIC WITH USER VALIDATION: The algorithm work as above but the algorithm also choose the plane that has the lowest centroid as the one most likely to be the ground plane. A window with the choosen plane is shown highlighted and on close the people detector is launched.
- FULL-AUTO: As above, but no window is shown and the people detection algorithm is launched as soon as a valid plane is returned.

Plane ranking works as follows: the algorithm is launched and the list of plane is returned, the ones with an inclination not compatible are ruled out. We assume 0 camera roll. So the planes with a b greater than 0.7 (Plane equation in Hesse normal form[4]) are ruled out. The remaining planes are sorted according to the centroid y, selecting the one with the highest value

[4]

$$ax + by + cz + d = 0$$

i.e. the lowest (camera model reference frame: y+ down, x+ right, z+ forward).

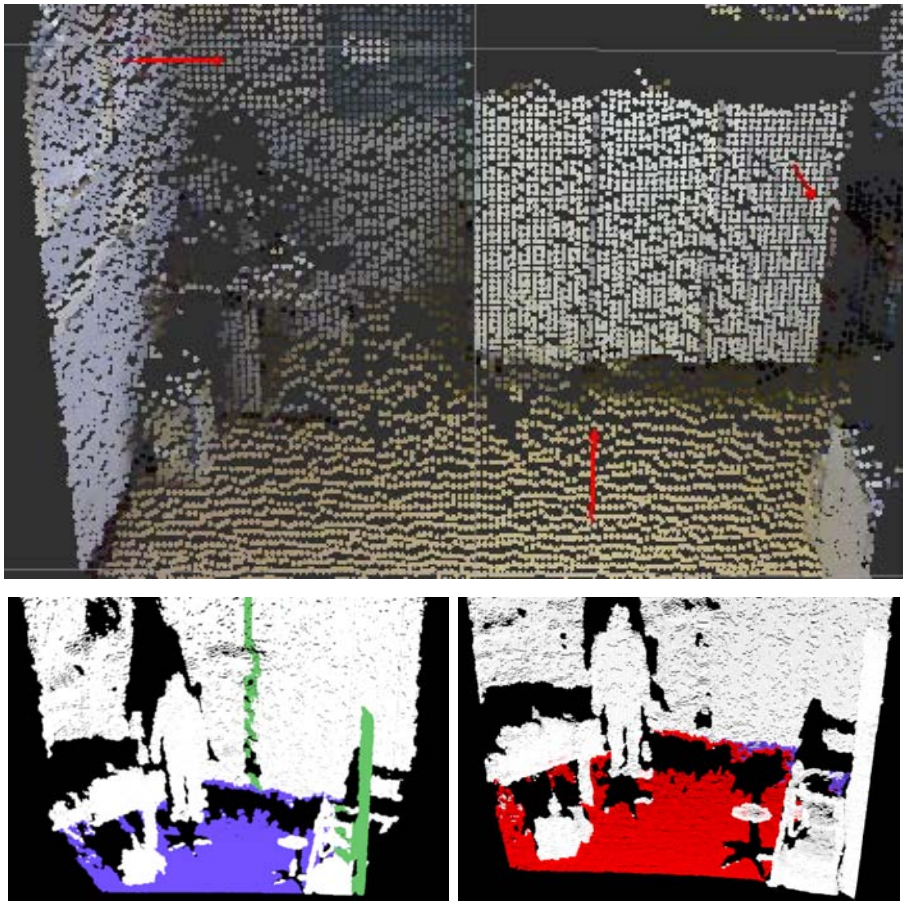

Figure 16: a) Connected Components Segmentation, red arrows show the direction of planar surface normal, the floor, the left wall and the separe' have been detected b) Semi-Automatic mode, user click once on the colored plane he wish to set the ground plane on c) Automatic selection with user validation: the application selected the plane colored in red, the user can check the plane is segmented correctly.

2.3 April tags

A fiducial marker is an object of known dimensions, placed in the field of view of an imaging system to be used as a reference for measures. Fiducials are an excellent method for pose estimation because they are easy to segment within an image and provide accuracy and speed, for this and other interesting perks they are pretty popular among the augmented reality community. Most of times, this type of marker has a binary ID encoded within it (lexicographic coding) and many times error checking bits are also present. A limitation of this system is the size of the dictionary of the possible strings representable with the structure of the marker (limited symbols). Designers must cope with the quandary between the size of the marker (in terms of bits per area) and the minimum distance between a marker ID to the next in terms of Hamming distance. AprilTags allows for use of markers of 16, 25, and 36 bits with respectively, with 5, 9 or 11 bits as minimum hamming distance. It's possible to generate other "families" of tags with custom size and distance, for this work the 36h11 family was used. It is downloadable from AprilTag's homepage. It's worth notice that we only tested the 36h11 family becauses it was "reccomended" by the authors, the question is still open if tags with less bits would be more recognizable from a distance compared to these.

Compared with other fiducial marker systems, AprilTags has a faster system for line segment computation, a tag encoding more robust and better accuracy in case of occlusions, lens distortions and lighting variation.

2.3.1 Marker Localization Pipeline:

- Preprocessing - The grayscale image from the Image Listener is normalized, that is, for each pixel, the integer value between [0-255] is converted in the corresponding decimal (floating point) between [0-1], and a gaussian smoothing operator is applied to reduce noise
- Gradients Computation - Magnitude and orientation of the gradient for every pixel is calculated
- Clustering - Pixels with similar direction and module of gradient are grouped in clusters, the clusters are cycled again to merge all segments lying on the same rect.
- Segment fitting - Segments orientation is calculated so that the dark side of the gradient is at the left.
- Quad Detection - Groups of segments forming a convex quadrangle are formed
- Pose Estimation - Through homography the quadrangle' s pose in respect to the camera is found.
- Tag Identification - The tag ID is decoded

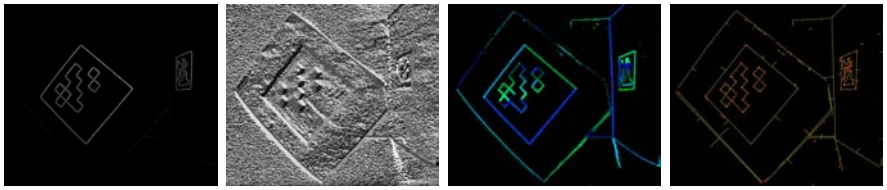

Figure 17: Tag Detection phases: a) Pixel gradients magnitude b) Gradients direction c) Clustering of pixels laying on the same rect d) Line segments inferred from the rects. The direction of the line segments is shown by short perpendicular notches at their midpoint.

Cuda SpeedUp Parallelizing some phases of the process with Cuda we achieved a substantial speedup in the tag extraction, without affecting the accuracy of the detection.

It was possible to parallelize the pre-processing phase (i.e. grayscale [0-255] to float [0-1] convertion and gaussian blur), the gradients computation and the edge cost computation.

In the first version of the software these two operation were been replaced with the cuda-enabled functions of the opencv library with a speedup of almost 10X compared to the original naive implementation of the specific phase. But because these require a custom-compiled version of the Opencv libraries their use has been appointed "optional". Instead, in the final release, we used the opencv standard versions with better performance to the naive version but not as good as the cuda-enabled version.

Custom kernels would allow to achieve performances equal or better than cuda-opencv's with the standard cuda library with no need to recompile the OpenCV library.

Gradient's magnitude and direction computation has been parallelized with an ex-novo kernel based on the original algorithm. Altough opencv provides these functionalities (Sobel Operator), the original formulation should guarantee better numerical stability and results more similar to the original CPU implementation.

Much effort has been put to parallelize the function that computes the edge costs for the clustering, being the most computationally expensive task

of the entire process, due to the nature of the task, it is difficult to speed up using memory coalesced accesses, but still we had better results than the CPU version even with a naive parallelization. Using cuda shared memory would save us a portion of the global memory slow accesses.

The union find algorithm is less expensive but still has a computational weight comparable to that of the edge cost computation. This algorithm has some critical sections and race conditions may occur but, altough non trivial, some solution to parallel union find exists in literature and can be applied to this problem.

Because AprilTags and OpenCV use different formats and conventions for raster manipulation, some work-around was necessary to handle memory occupancy and limit the conversion and upload (to/from video memory) overhead. Image rasters, (the raw pixels values) can be encapsulated ("wrapped") with any number of different "headers" providing interface to their library respective algorithms with no need to create a whole other object in memory that is indeed equivalent to the other.

cudaOps class Every function has a standalone version that could be used independently from the others. To optimize memory transfers, alternative version with the suffix "Opt" has been written that save deallocating video memory and allowing the next function to use those locations. Needless to say, these must be used all or none.

- CONVERT: The image pixels are converted from grayscale integers [0-255] to decimal floating point values [0-1]. We tried in order, the opencv-cuda method "convertTo", than C++ STL method "transform", and, lastly, the standard opencv "convertTo", and kept this last one.

- GAUSSIANSMOOTH: first we used the opencv-cuda version, than we switch to the standard opencv version.

- COMPUTE_GRADIENTS: The per-pixel operations are the same, moving the operation on the gpu allowed us to gain a larger bandwidth exploiting memory coalesced accesses[5].

- EXTRACTEDGES: computes the four costs of linking every pixel with four of its neighbours (upper, right, upward-right, downward-right). Future implementations could exploit on chip cache to load the working image "tile"[6] on the ludicrously faster on-chip memory and reduce global memory accesses.

Results Compared to the old cpu-only implementation our approach resulted in a more reactive cube tracking and also we exempt the cpu from a quite heavy portion of computing, making the other modules run substantially more fluent.

[5] In cuda programming a *coalesced* memory access is such that threads with consecutive Ids access simultaneously to consecutive memory locations (not all threads need to partecipate) this practice is very important when porgramming with cuda and must be always kept in mind.

[6] Cuda kernels launch threads in blocks of up to three dimensions, this is to provide a further level of abstraction when designing parallel algorithms. It is reasonable, while working with images in Cuda, to have this blocks in a rectangular/square form.

#nodes running/ex. time	GPU	CPU	SpeedUp
1 node	0.38	1.30	~3.4X
2 nodes	0.50	1.60	~3.2X

Table 1: Comparison of execution times between the old cpu-only version and the cuda-enabled version on FullHD 1920x1080 pixels images. We found out an average speedup of ~3X that decrease sublinearly when running multiple nodes.

2.3.2 Sensor Fusion and Kalman Filtering

Sensor Fusion is done combining noisy detections from different sensors to get an estimate of a phisical property we desire to measure (in our case the marker's position) with less uncertainty than using one single sensor.

The Kalman filter algorithm uses a system's dynamic model (laws of motion), known control inputs to that system, and multiple sequential measurements to infer the system varying quantities (state) better than the estimate obtained by using any measurement alone would do. The Kalman filter averages a prediction of a system's state with a new measurement using a weighted average. The purpose of the weights is that values with better (i.e., smaller) estimated uncertainty are "trusted" more while values with larger uncertainty have less effect on the average value. The weights are calculated from the covariance, a measure of the estimated uncertainty of the prediction of the system's state.

As a consequence, varying the covariance value introduces a short blanket problem, whit a low value the resulting average over time will be "smoother" but also less "reactive" meaning that the delay between the actual changing of the property to measure and effective measurement will increase. While increasing covariance will decrease this delay, but also bring some of the uncertainty of the measurement in the final value.

In our implementation, the covariance value for centers fusion is slightly higher than the value for tags fusion, because most of the uncertainty in detection has already been filtered in the first stage, so we can use a more reactive fusion when merging centers detection.

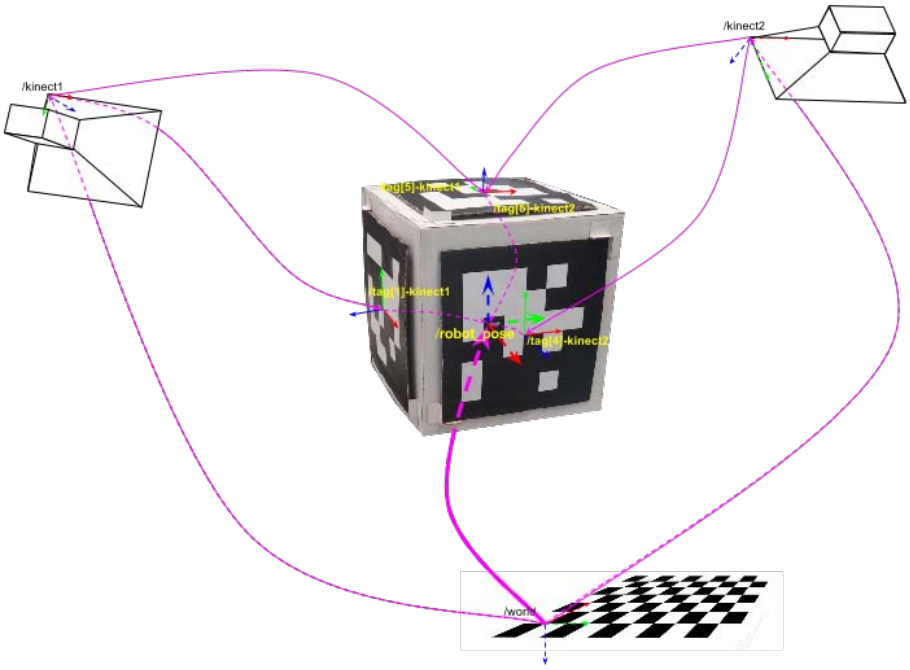

Figure 18: Center Pose Fusion: the cameras' positions are registered to the world frame so we can get the pose of the markers w.r.t. the absolute reference frame. In this example the first kinect sees the tag[1], the second kinect sees the tag[4] and both sees the tag[5], however, we have less uncertainty on the last one because we can achieve sensor fusion with multiple sensors sampling the same tag. The projected centers are already in the world frame, a second Kalman Filter fuses these in the whole marker's pose.

2.3.3 Composite Marker Detection, cubic marker

To make the robot traceable in a camera network, from every possible point of view, we used the AprilTags in a particular cubic set up. We placed 5 different markers on a cube of approx. 30cm3 , one for each of the 4 walls and one for the roof, so that the robot pose can be determined from every camera all around and over the robot. The origin of the robot_frame is the exact center of the cube, geometrically calculated from the center of any face with an offset of z that shift it inwards the cube. This generated

frame is called projected center or virtual center. To have a more stable tracking of the center of the cube, cube same-face detections among two or more cameras are fused by a kalman filter. So that in the scene we can have an absolute position w.r.t. the world frame of reference for any face/tag. A similar fusion is done for the projected centers of the cube. If more than one face is visible by the cameras, than we have more than one projected center, another kalman filter tuned for this task will fuse all the detections of the virtual centers into one that is the actual cube center.

2.4 PCL::People vs GroundHOG

2.4.1 cudaHOG-Based detector

The standalone version of the RGB detector should theoretically have a wider detection range than those based on a structured light sensor, as the depth sensor accuracy and functionality depends on the distance and it is reliable only under a certain threshold. The RGB image allows to detect people even at long distances, the only limits are sensor resolution and the sampling precision specified for window generation. During the tests, we discovered that cudaHOG can detect people further from the camera than PCL::People, the only limit is the hog_start_scale. This paramenter determines the minimum window size to be generated and evaluated by the svm. However, we need the point cloud to have a 3d correspondence for the 2d bounding box returned by cudaHOG, as we exploit the organization of the point cloud. This furtherly constraints the search radius if, for example, we use kinect-like sensors instead of a stereo pair. However it is worth notice that with this system we need less spatial resolution in the point cloud, because instead of using it for evaluating the svm, our approach exploits the point cloud and the depth information for a "validation" of already 2d-evaluated bounding boxes.

Altough the hybrid detector has proven functional in detecting people, we didn't use it in our tests with the robot because detecting/tracking/following people not in contact with the ground was not in the scope of this experiment. Also because the ground based people detector process is much more light-weight than the cudaHOG based detector and it's indeed a better candidate to run in a multi-process context. Furthermore, we are already using the Gpu for tag extraction.

To build a 3d bounding box compatible with openPTrack from cudaHOG 2d bounding box we had to apply a few simple geometric transformation. We are working with a registered and organized point cloud so, there is a one-to-one correspondence between the pixels of the RGB image and the points of the point cloud. The cloud points are indexable with a width and an height as those of the original 2d image. As a consequence we can get

the corresponding 3d point in a point cloud from any point in a 2d image in constant time. To generate the bounding box we proceed as follows.

First we compute the centroid of the window, as $(p1.x + width/2, p1.y + height/2)$ and we get the corresponding 3d point in the cloud in O(1). That point will with all probability lie near the person centroid, to compensate for the person thickness we add 10 cm to the Z coordinate of the point, we call this point P0 (Person Centroid). Than we compute the midpoint of the upper edge of the 2d BB (p1.x + width/2, p1.y + 10) adding 10 pixels to the y so that we take the head centroid more accurately and get the corresponding point in the cloud, we call this point P1 (Person Top). The last point P2 (Person Bottom) is computed as 3D reflection of P0 with P1 with the formula:

$$P_2 = 2 * P_0 - P_1$$

In one of the first implementations of the algorithm the process was inverted, the person top was computed as reflection of the person bottom with the person centroid. During testing, we saw that the bottom centroid was not easily detected in some cases, so we decided to go the other way around. In this manner, we managed to have the system to work even in case of partial occlusions (waist-down).

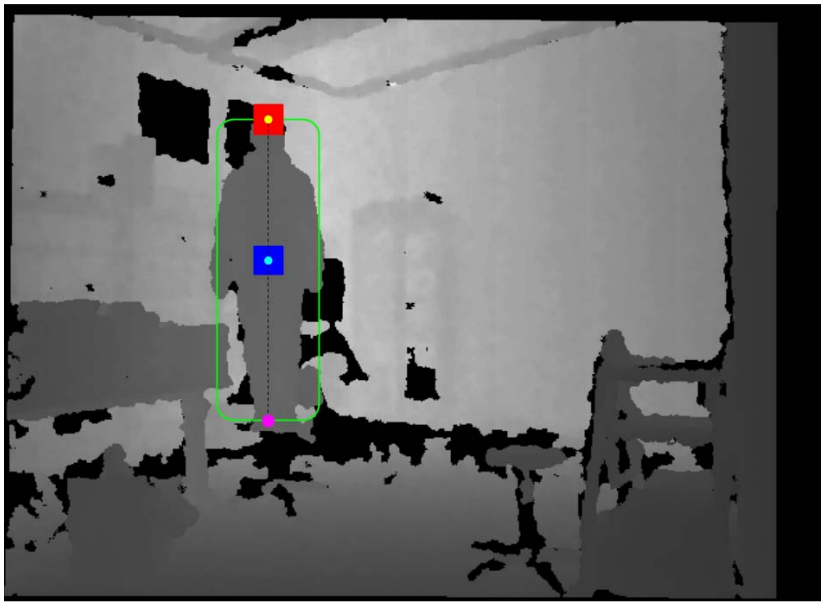

Figure 19: *To integrate the new detector in openPTrack it was necessary to convert the 2D detection from cudaHOG to the format the HaarDispAda node takes in input (3D). This to avoid braeking the cascade of detectors. The detection window (in green) is used to find the coordinates int the 3d space. Person centroid is in light blue, head centroid is the yellow dot, P2 is the magenta dot. The squares represent the searchAround() area.*

searchAround function During testing we detected another problem. We saw that sometimes, when trying to resolve the 2D->3D point correspondence from image to pointcloud, these were not set to some value of X, Y, Z, but to *NaN*.

NaN is a numeric data type value representing and undefined or unrepresentable value, mainly are pixels for which the kinect driver couldn't find a feasible value for depth and consequently failed 3d reprojection.

We had to devise a strategy to search "around" any 3d point in the cloud for other possible candidates, and select among these one suitable as head or person centroid. Given a 2d point (x, y) of which we desire to get the corresponding 3d point (X, Y, Z), searchAround() looks for all the valid (not

NaN) points in an area around x,y and selects the median, instead of the average, of Z, so that the outliers do not affect the final value.

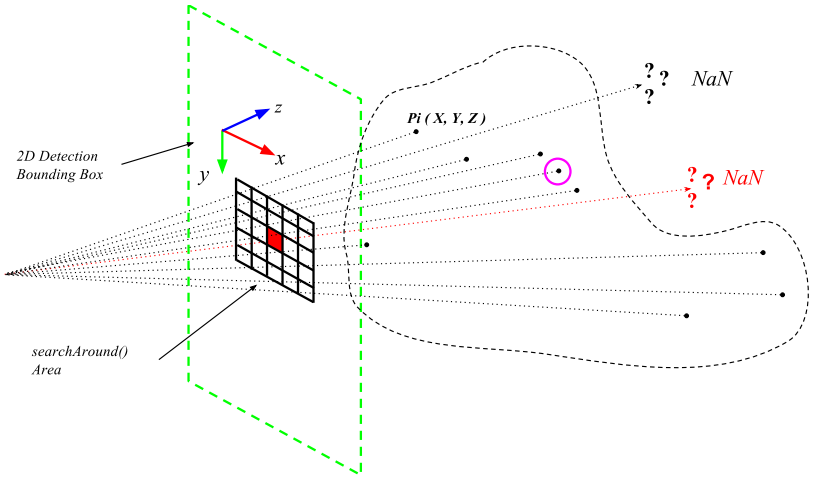

Figure 20: *The searchAround(x,y) function looks for a valid cloud point that is near the 2d point given as input, exploiting the registration of the point cloud with the analogue 2d image so that we don't have to run a nearest neighbour search in the 3d space.*

structure of the package: A package containing the main program, cudaHOG libraries and the svm model for pedestrian detection created with SVMdense from the INRIA person dataset.

The point cloud and rgb image topic must be specified, optionally *hog_start_scale* and *hog_scale_step* can be specified.

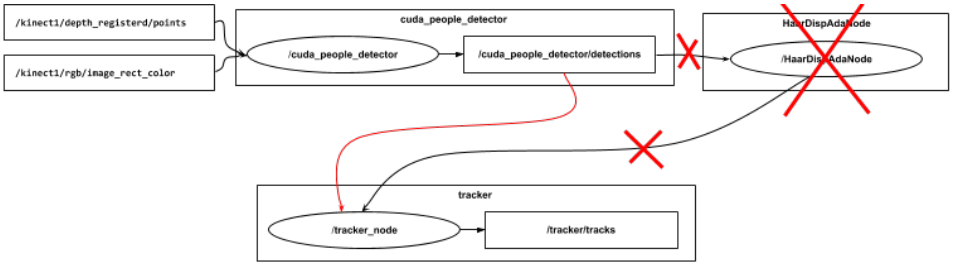

Figure 21: Modifiche alla struttura dei nodi

performance analisys:

Detector performance (with/without HaarDispAda) Now we will compare the performance of the two detectors in a simple use case, with only one person to track.

Figure 22: Frames from testing scenario

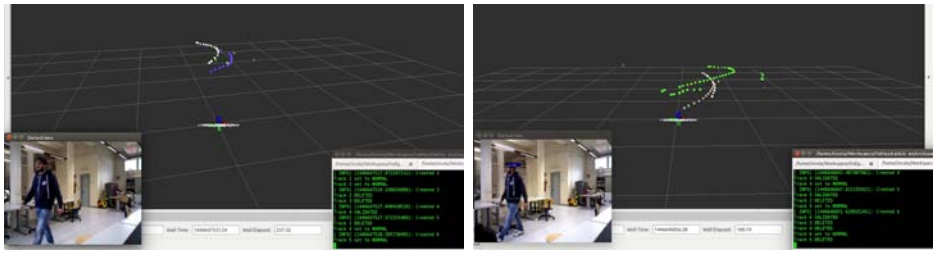

Figure 23: On the left we can see the tracks produced by placing the cudaHOG-based hybrid detector upstream the HaarDispAda node, so that the tracker is fed with detection filtered by the latter. On the right we see the results of connecting the cudaHOG detector directly to the tracker.

By a simple visual analisys we can notice that the tracks produced by cudaHOG+HDA are more compact because HDA filtered many of the detection that cudaHOG let pass (false negatives), hence, we deduce that in this configuration it would be advisable to tune HDA minimum confidence to avoid this phenomenon.

On the other hand, de-activating the HDA node we noticed an higher occurrence of false positives, that returned longer tracks but more fragmented, because the tracker re-association has failed.

Both are valid solutions, once the detector/tracker parameters are tuned up for the task, the additional check performed by HDA, would filter the (few) false positives of thre cudaHOG module and return precise and reliable detections.

Detector performance (vs PCL::People) Leaving out the pros of cudaHOG we already discussed, on a frame-per-second metric the hybrid detector can't compete with PCL::People on the testing machine. Even with little demanding values for the detector paramenters *hog_start_scale* and *hog_scale_step*, to allow for a decent detection speed with a comparable performance (quality of detection).

We scored a detection speed between 10-15 sec, altough these are quite sufficient for the task and cover the computational capacity designed are very far from the 60 fps of PCL::People.

It's worth notice that this can be an unfair trial, since cudaHOG runs on GPU and PCL::Peole runs on CPU, it would be interesting to confront the two using a better performance video adapter.

Tracker performance PCL::People uses two a cascade of two detectors, because the clustering phase generates many detection windows that are passed to the HDA node to be filtered and have reliable detections. Initially, the cudaHOG node was put upstream HDA node with the latter sending filtered detections to the tracker, with fair results. The tracker has some parameters that specify the condition for the creation of a new track (e.g. minimum number of consecutive detections, minimun confidence, etc.). If the detections are more sporadic, the initialization will be delayed, the track will be made of less points, hence less accurate. Worst case scenario, track re-association fails and a new track is created for the same person. So it's important that detection is continous. Connecting the tracker to the cudaHOG detector gave us a noticeable increment in quality and continuity of the tracks and a decrease in the number of false positives.

2.5 Nodes structure

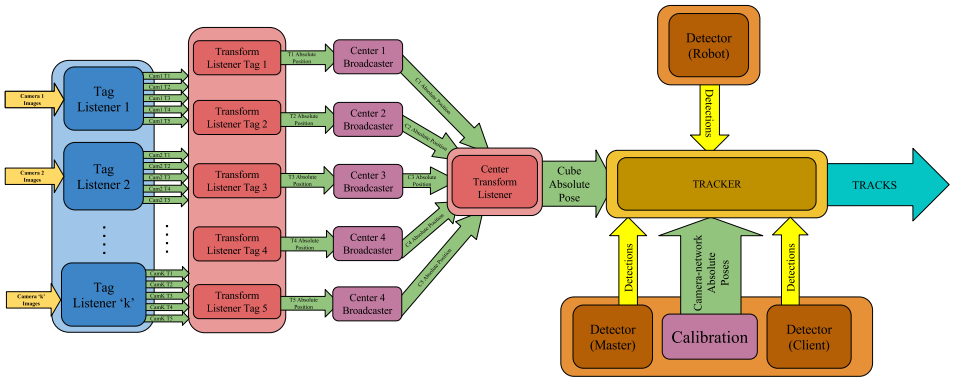

Figure 24: Nodes Structure: the cube tracking node continuosly update the current position of the marker in the tf tree

Apart from bag playback, we should spend a few words on the optimal set up that this kind of system should use. The ideal load distribution for any single machines should divide efficiently the workload between the master and the clients but also keep network occupancy contained so that the total bandwidth of the topics transmitted over the local are network (LAN) never exceeds the maximum network bandwidth. Assuming Master as the most powerful machine in the ROS network:

- As Client Node: A client node should run a People Detector and a Tag Listener for any of the cameras attached
- As Master Node: As above, but also the tracking node and transform listeners nodes (tags and center) should run here.

The idea behind this is to avoid transmitting large messages like FullHD images and point clouds over the LAN, instead send only tf's and detection's messages that require a much smaller channel capacity.

3 Setup, Testing and Experimental Results

During our experiments we noticed that the tracks produced by the camera robots were affected by the camera motion. The problem is particularly evident in case of camera pan, that occurs when the robot rotates on his vertical axis. The followed person appears to "move" inside the picture, while in reality is just a visual effect due to the camera pan. However this effect is reported to the tracker as a person motion, affecting the quality of the tracks produced. In fact, as the marker-robot-camera system moves, the relative position of the camera w.r.t. world frame has still to be updated while the detector is sending inconsistent detections. This is caused by the difference in respose time between the people detector and the cube localization routine. This effect worsen with increasing distance from the camera.

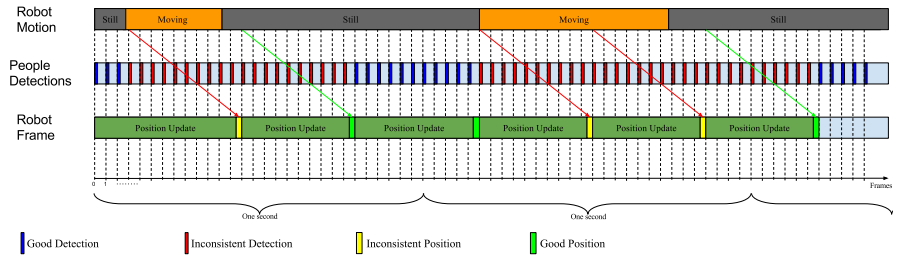

Figure 25: The timeline above shows the inconsistency problem relative to the robot detector, the "Robot Motion" timeline represents the state of the robot, Moving/Still (for the sake of simplicity, we just considered the case of the robot rotating as it's the most problematic). The second line represents the People Detector that runs on the robot, in this case set to run at 30Hz. The last line represents the marker tracking routine, the position returned will be inconsistent if the robot changes position during the update.

As we said, a detection is always referred to a particular reference frame. Let's assume a consistent position at system start. While the robot is still not moving everything works as it should. Then the robot starts moving, the cube tracker grabs the frame of the robot in motion and starts processing the frame to know the robot's position in this instant. Meanwhile, the people detector keeps iterating, referring these detection to the old reference frame.

When the robot stops and the tag processing returns the tag position from a frame with the moving robot, this is inconsistent, because it's referred to a previous video frame and instant with the robot still in motion. If, while the robot is not moving, a frame is grabbed and the robots remains still until the tag tracking routine returns its position, this will be consistent with the actual position of its camera and the people detector will return consistent detections.

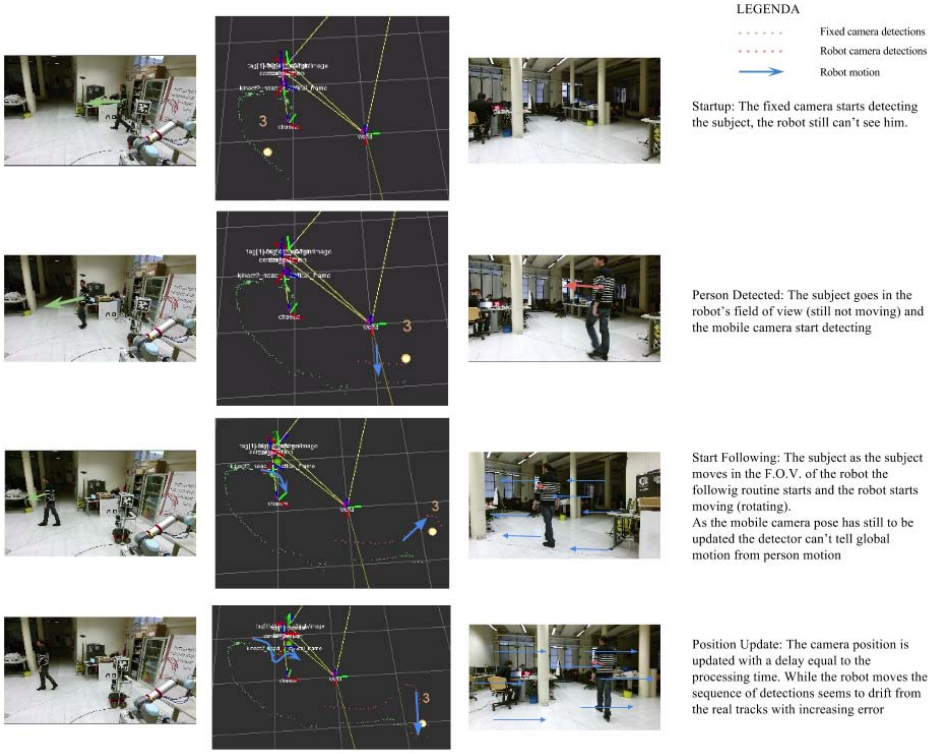

Figure 26: In these sequence of pictures we can visualize how the tracks are affected by the responsiveness of the localization. The delay in the camera registration makes the people tracker interpret global motion as people motion while the system has an inconsistent fix on the position.

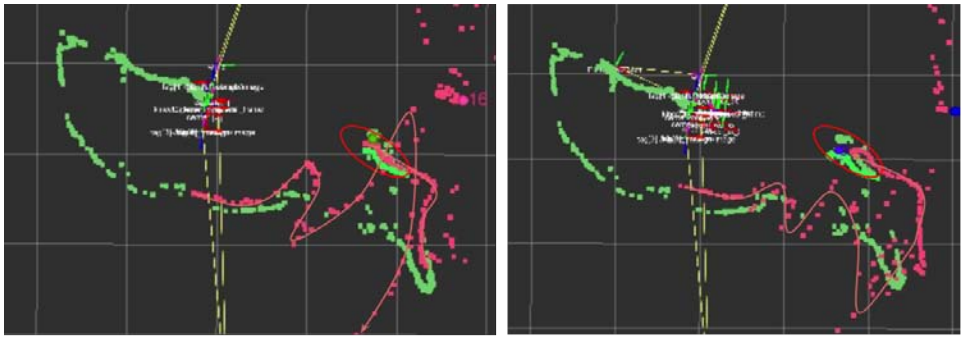

Figure 27: Left: case *without using* odometry. Right: case *using* odometry.

Detection Quality Once the robot has stopped and a consistent fix on the camera position is returned the detections are correctly reported. The resulting detections in the second case has less pronounced meanders. As we can see in the red ellipse, when the tracks reconnect, the robot detections are a little back confronted to the fixed cameras' due to wheel odometry drift.

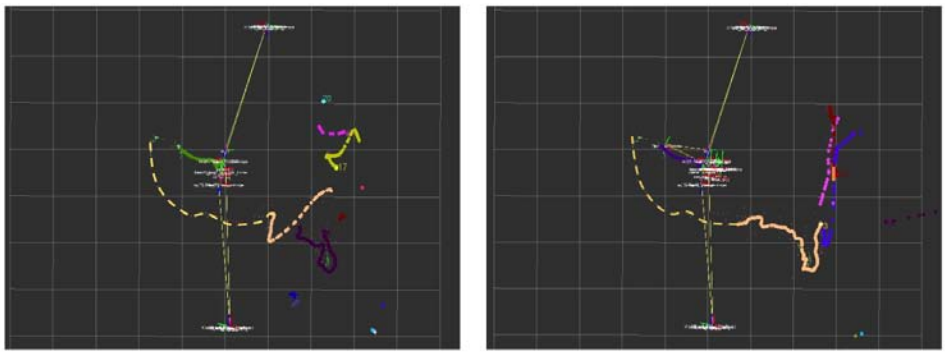

Figure 28: Left: case *without using* odometry. Right: case *using odometry*. (The initial part of the track has been added manually for completeness)

Tracking quality As we can see, in the first case, the delay between detection and repositioning, causes the tracker to lose track of the person when the robot rotation is pronounced and the person is far from the robot camera (the person's track is initially beige, then his track is re-initialized in purple). On the other hand, using odometry, we have one single track for all

the duration of the experiment (note: the other small tracks are noise due to a lowering of initialization confidence).

First work-arounds Diminishing the rate of the robot detections per second gave sligthly better results, because less inconsistent detections are sent to the tracker while a valid position is not available. However, this drastic remedy largely affected the completeness of the tracks as less detections are sent to the tracker, so we rule out this idea.

The optimizations that were made to the AprilTags C++ interface to reduce tag pose extraction process execution time, gave the routine better responsiveness. Some of the phases have been sped up using Cuda, pointer arithmetic, C++ Standard Template Library methods, and OpenCV optimized API, the last one in particular for image normalization and gaussian smoothing. But still not fast enough to solve the problem. For the system to be totally consistent tag detection for a single frame should always have a smaller latency/execution time than people detector's.

Wheel Odometry Wheel odometry (the trajectory that is computed from the phisical design and modelization of the robot drivetrain and the input velocities given to the wheels) could be used to aid the mobile camera tracking. Wheel odometry is considerably faster to compute than the visual marker position, so its use would partially solve the inconsistency problem. However, the wheel odometry trajectory has the tendency to drift from the real value as the robot moves along its path over time. Differently from tag detection that has a zero mean error over time. Furthermore, the accuracy on the measurement of a single rotation depends only on the extent of the rotation. Hence, we could exploit wheel odometry information to adjust camera position during rotation, altough translational drift should be taken in to account.

We achieved that by creating a new node called odom_refinement, that works as follows:

1. At startup, the cube is detected and the pose is computed, we save the transformation.

2. Then, we assume that the robot odometry origin on the ground, vertically aligned with the cube center.

3. Finally, we registrer the odometry origin with the world according to the transform previously saved, keeping broadcasting it over the whole experimet.

This approach has proven to give better results, as we have less disturbance in the mobile camera people detections due to inconsistent detection. But we traded a more precise tracking system for one that is more reactive but less accurate over time. The ideal configuration would allow us to exploit both systems in a more dynamic and integrated manner. For example, if we are not interested in the whole robot path, we can reset the odometry origin at regular intervals and update its pose w.r.t. the world accordingly to the robot position at the moment of the update (making sure the robot is not moving in the process).

Or, otherwise, correct the odometry vector (from origin to robot base link).

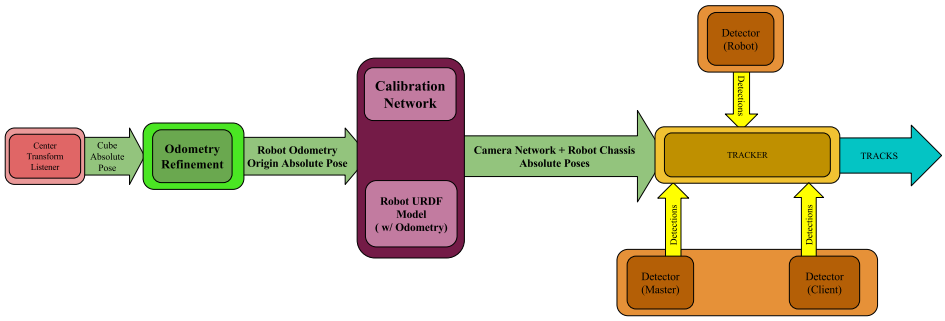

Figure 29: The node's structure when using wheel odometry: At startup the cube listerner sends the robot/marker pose to the Odometry Refinement node, this will link the robot odometry origin to the world frame. From now on wheel odometry information is part of the same camera network tree as the fixed cameras.

4 Conclusions and future developments

In this thesis we used a omnidirectional visual marker to track a robot while moving, and on the same time, we used this mobile platform to do people detection. The calibration software correctly refer the fixed cameras in the system to a common reference frame, allowing distributed detection and tracking.

The april tag listener has gone under heavy code maintenance and tuning to optimize at best the detection speed, further optimization is possible would be possible using SIMD instructions and parallelization also in other computationally expensive phases of the detection. Edges merging did benefit from this, altough the parallelization (expecially in cuda) of the Union Find algorithm is not trivial. With a well designed parallel algorithm it would be possible to reduce computing time for the U.F. phase from ~100ms by a factor of 3 if the same trend of the other phases holds.

The two level sensor fusion routine manages to track accurately the cubic marker, although the processing time required to compute a single frame affects the quality of the tracks produced by the people detection algorithm. Using wheel odometry instead of tag tracker gave better results, but still a routine to compensate wheel odometry drift would be required.

The detection node has been enhanced to work in presence of elevated sensor tilt without affecting the speed or quality of people detection as the point cloud rotation run efficiently in linear time.

The ground detector can extract the ground equation rapidly and accurately so that is available to the ground based people detector. The four modes of the ground estimation module, now standard in OpenPTrack, cover most of the use cases that can occur. The automatic mode is the default, in the distant case the selected plane is not the one desired, two manual selection modes are available, and the debug mode can show the selected plane to the user to validate before launching the detector.

With the hybrid detector the 2d detections inferred from the rgb image are transposed in the 3d coordiante frame using image to point cloud registration, and a probabilistic method handles bad correspondences. Also, now

people not in direct contact with the ground can be detected and tracked, altough not in real time.

In conclusion, the experimented methodology holds, because if the robot is still and the fix on the position is consistent, the dinamic registration of the marker can refer the new detection to the new position of the robot camera. However, the only way to have the system run in real time, in such a way that it is completely and always consistent, without using wheel odometry, is to have the tag detection run faster than the people detection.

4.0.1 Future developments

Marker Tracking To furtherly speedup detection we can use the assumption that only one robot is always in the scene. Instead of scanning the whole image for markers we can restrict the search area to those pixel that in a previous frame contained the marker. The detector will track the area containing the marker and ignore the outer pixels. A whole scan would be done only if no tag has been detected at the previous iteration and/or at regular intervals. This way we would generate less edges, and lighten the workload for all the successive steps of detection.

Detector suppression Another possible solution would resemble what happens in the human brain during eye movement. *Saccadic Masking*, also known as (visual) saccadic suppression, is the phenomenon in visual perception where the brain selectively blocks visual processing during eye movements in such a way that neither the motion of the eye (and subsequent motion blur of the image) nor the gap in visual perception is noticeable to the viewer. So, we could, for example, re-design the system this way: when the wheel driver issue a rotation order (the most problematic case) a signal is sent to the robot's people detector to pause detection. It will be restarted only when the robot has already stopped AND the system has a consitent fix on the marker position (i.e. the tag listener grab a frame and return the position *all* while the robot is not moving).

Visual Odometry Visual Odometry in robotics is the process of incrementally estimating the pose of a vehicle by examining the changes that motion induces on the images of its onboard cameras. It has been used in various robotic application, among other things also the Mars Exploration Rovers. Briefly, is a feature based method: "interesting" points are tracked between two sequential images that frame the same scene, then the transformation that brings those points from the first image to the second is estimated and used to compute the robot new pose w.r.t. the first image. The solution

is found by determining the transformation that minimizes the reprojection error of the triangulated points in each image.

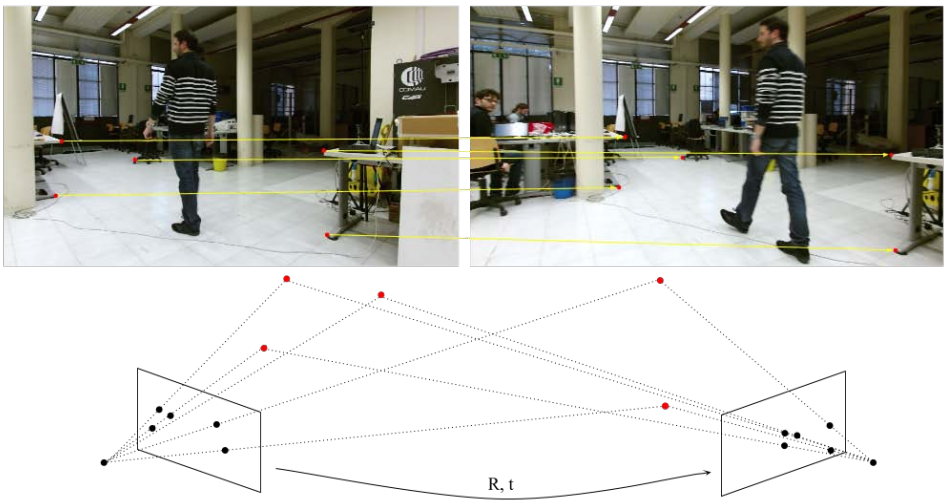

Figure 30: Visual Odometry 2D-2D, five points is the minimal case solution, using 3D data (e.g. from the point cloud) just 3 points are needed

The basic case, i.e. 2d to 2d V.O. needs a minimum of 5 good points to make a univocal solution. Using 3d data from kinects 3 non-collinear points are sufficient.

Here is a possible workflow, we exploit PCL's registration APIs:

1. Start marker and people detector
2. When the robot moves start visual odometry:
   (a) Grab frame
   (b) Segment (remove) people
   (c) Register the cloud at frame n+1 with frame n and get transform
3. Use the computed transform on the old reference frame and update it.

References

[1] P. Sudowe, B. Leibe: "Efficient Use of Geometric Constraints for Sliding-Window Object Detection in Video", UMIC Research Centre RWTH Aachen University, Germany

[2] A. J. B. Trevor, S. Gedikli, R. B. Rusu, H. I. Christensen: "Efficient Organized Point Cloud Segmentation with Connected Components"

[3] M. Munaro, A. Horn, R. Illum, J. Burke, R. B. Rusu: "OpenPTrack: People Tracking for Heterogeneous Networks of Color-Depth Cameras", Center for Research in Engineering, Media and Performance, University of California Los Angeles, Los Angeles, CA 90095-1622, USA

[4] N. Dalal, B. Triggs: "Histograms of Oriented Gradients for Human Detection INRIA" Rhone-Alps, 655 avenue de l'Europe, Montbonnot 38334, France

[5] M. Munaro, E. Menegatti: "Fast RGB-D people tracking for service robots" Springer Science+Business Media, New York 2014

[6] ROS, Ros Operating System. Framework open source usato per comandare e ricevere dati dai robot, sito http://www.ros.org/

[7] OpenPTrack, software open source che mira ad offrire uno strumento scalare, multi immagine per il tracking di persone, sito http://openptrack.org/

[8] NVIDIA Corporation, NVIDIA CUDA C Programming Guide, October 2010, 2701 San Tomas Expressway Santa Clara, CA 95050.

[9] M. Munaro, A. Horn, R. Illum, J. Burke, R. B. Rusu: "OpenPTrack: People Tracking for Heterogeneous Networks of Color-Depth Cameras", Center for Research in Engineering, Media and Performance, University of California Los Angeles, Los Angeles, CA 90095-1622, USA

[10] *Confronto tra marker Aruco e April Tags per la stima della posizione di un robot mobile all'interno di una rete di sensori Kinect*, L. Lazzarini

[11] *April Tags, a Visual Fiducial System*, http://april.eecs.umich.edu/wiki/index.php/AprilTags

[12] Multimarker tracking, ARToolKit documentation: http://artoolkit.org/documentation/doku.php?id=3_Marker_Training:marker_multi