Autonomous Navigation of Nao using Kinect
CS365 : Project Report

Samyak Daga
11633
samyakd@iitk.ac.in
Dept. of CSE

Harshad Sawhney
11297
harshads@iitk.ac.in
Dept. of CSE

Indian Institute of Technology, Kanpur

April 21, 2014

Abstract

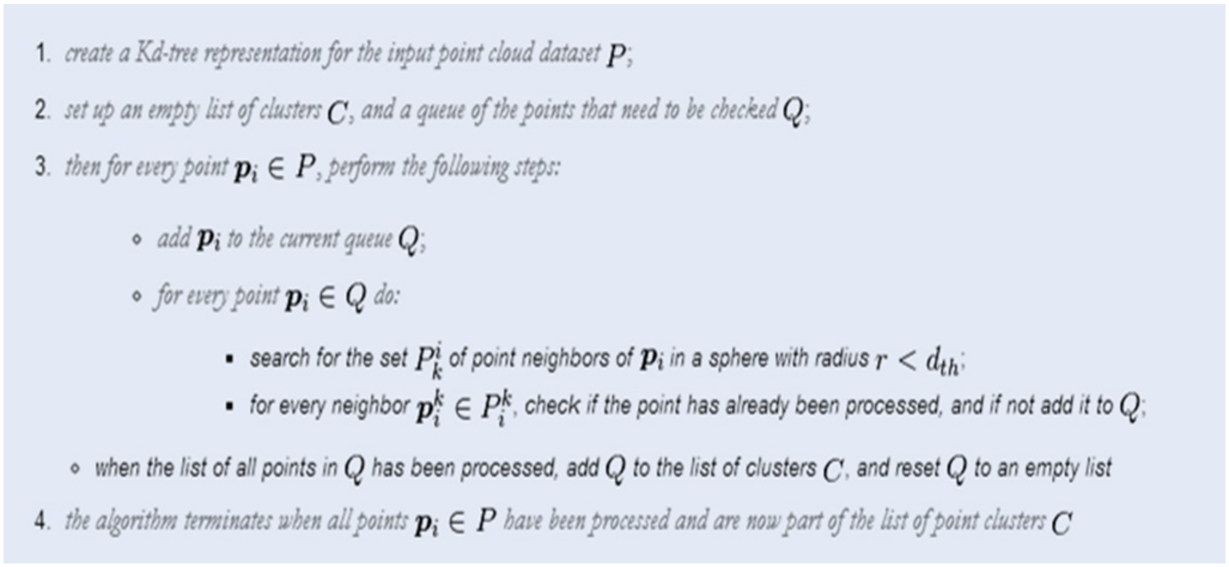
The navigation of humanoid robots is a non-trivial task with widespread applications ranging from assisting humans in their daily chores to manning space missions to disaster recovery operations. We present a depth based sensing approach for autonomous navigation of humanoid Nao. The objective is to reach a user desired goal position from a source position using the data of depth based sensor for visual servoing. The localization of the robot is done via Monte Carlo techniques which uses the RGBD data of Kinect sensor. The cluster of robot is extracted from the environment followed by KLD (Kullback-Leibler distance) sampling to accurately track the robot in real time. The motion control of Nao relies on the feedback data obtained from the depth sensor to correct its path which could possibly deviate due to noises in odometry and sensing information.

1 Introduction

Autonomous navigation of humanoid robots has been a subject of widespread and lively research over the last years. The problem is highly challenging and non-trivial, with heavy potential impact in real world applications and usage.
Inaccurate foot odometry, noisy overboard sensor information caused due to the swaying motion of the humanoid, limitation in mechanical movements and joints, and the constraints on the degrees of freedom are some of the problems associated with the navigation of the humanoid. These issues lead to the deviation of the humanoid from its expected path, resulting in not reaching the desired goal.
Our project tackles the problem of navigation towards a desired goal position. This has been done in context of developing an assistive humanoid robot for home environments for helping the elderly and picking garbage from the floor.
Our system uses external sensing techniques, in particular the depth data, for the control of movement. We use the Kinect data sensor to track and detect the humanoid. Xbox Kinect is a low-cost sensor equipped with an infrared laser which provides a point cloud of the environment and gives the RGBD data of each such point. The point cloud is used to track the humanoid in real time.
The humanoid that we employ for testing our approach is Aldebaran Nao. It is a biped robot with about 25 degrees of freedom, and a human look alike.

1.1 Related Material

Autonomous navigation has been a problem which has seen a lot of different methods. Laser sensors, 2D cameras, depth cameras have all been employed with significant variations.
Corridor navigation through the use of 2D cameras has been used by Oriollo et al. [3] with the concept of vanishing point.
In [2] an external Kinect is used to track the 6DOF pose of Nao, which is then used to navigate to a goal position serving as an assistant in home-like environments Kinect has also been mounted on Nao itself by Maeir et al. in [5] which is then used to detect obstacles and plan the path by the construction of a heightmap KLD sampling technique proposed in [4] was used in [4], along with Monte Carlo localization, for tracking objects in a point cloud Monte Carlo localization was proposed in [6].



```
1. create a Kd-tree representation for the input point cloud dataset $P$;
2. set up an empty list of clusters $C$, and a queue of the points that need to be checked $Q$;
3. then for every point $\mathbf{p_i} \in P$, perform the following steps:
        ◦ add $\mathbf{p_i}$ to the current queue $Q$;
        ◦ for every point $\mathbf{p_i} \in Q$ do:
                ▪ search for the set $P_k^i$ of point neighbors of $\mathbf{p_i}$ in a sphere with radius $r < d_{th}$;
                ▪ for every neighbor $\mathbf{p}_i^k \in P_i^k$, check if the point has already been processed, and if not add it to $Q$;
    ◦ when the list of all points in $Q$ has been processed, add $Q$ to the list of clusters $C$, and reset $Q$ to an empty list
4. the algorithm terminates when all points $\mathbf{p_i} \in P$ have been processed and are now part of the list of point clusters $C$
```

Figure 1: The Euclidean segmentation algorithm [1].

2 Approach

2.1 Euclidean Cluster Segmentation

A cluster representing Nao is obtained from the point cloud obtained from Kinect using Euclidean segmentation. The motivation behind it is to reduce the processing time for point cloud P by dividing the unorganized data using a clustering method based on Euclidean distance metric. The algorithm is shown in Figure 1 [1].

A K-d tree format is used for storing the point cloud which helps in efficiently storing the point cloud P. A K-d tree representation stores the points in a binary tree format where the centroid of all the particles in its domain is used to split the point cloud at every depth level. Now, each point in the point cloud can be accessed quickly, thus saving time.
Then, each point is taken from the point cloud and a sphere of radius r is taken around and all those points lying in the sphere are placed in the same cluster. Then, for every such neighbour added to the cluster, the same process is repeated until no more points can be added. This creates a cluster and by the same method, all possible clusters can be extracted by using the entire point cloud data.

Now, we have obtained the cluster of robot as shown in Figure 4. All the obstacles present in the image are also clustered and their centroids are obtained through the point cloud data.

2.2 Tracking

After, we extract the point cloud of Nao robot, the next task is to track the robot movement in real time motion. We cannot cluster the robot again and agian by the above process as it is too slow and real time motion could not be achieved.
We use particle filter or Monte Carlo localization technique to track the robot and KLD sampling is used for sampling the data in the space. We cannot solely depend on initial data and make the robot move to desired location as the walking motion of robot is error prone which will keep on accumulating over time.

2.2.1 Particle Filter

It is based on the assumption that the current state is dependent only on the previous state and requires a static environment. Each particle is considered as a possible state. It consists of three steps which are shown in the Figure 2.

1. Motion Update:
   The particles are shifted according to the change in position of the point cloud. This is predicted using the approximate point cloud coherence which predicts the motion of the robot.

2. Sensor Update:
   Now, the Kinect sensor provides the estimate of the robot. This is compared with the predicted position of the sample particles in the previous step. The sam-

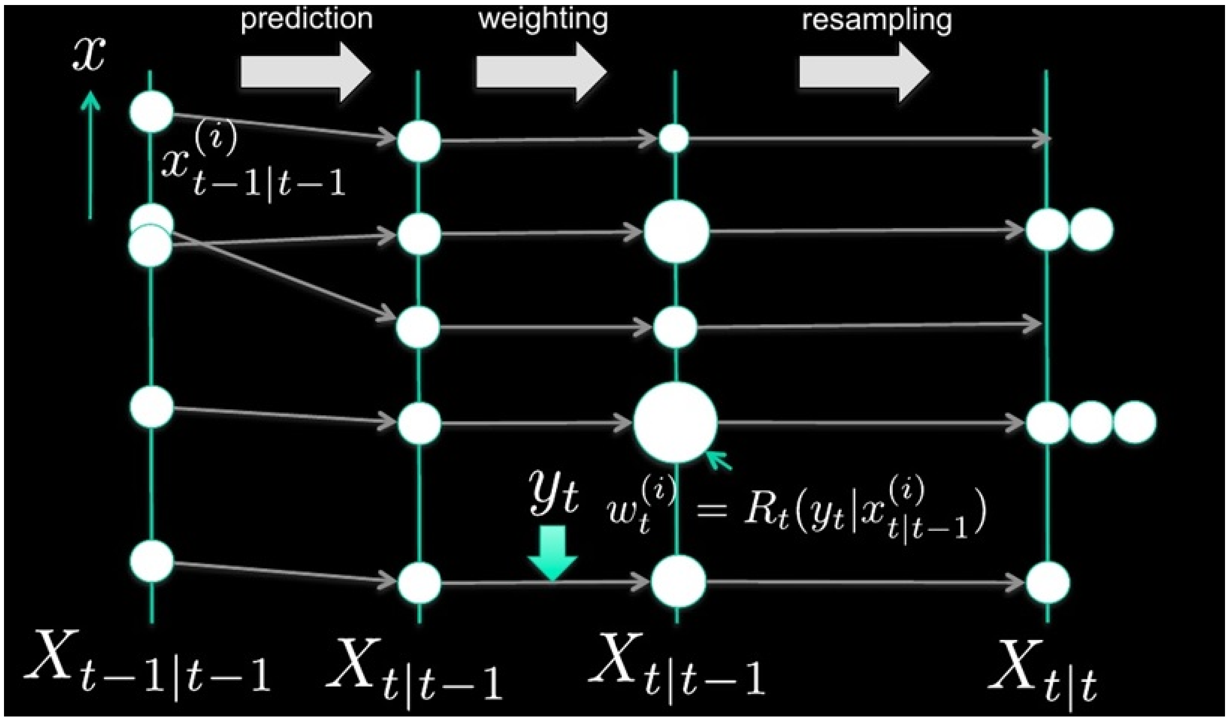


Figure 2: Description of the methodology of particle filter [6].

ple particles are provided weight on the basis of the closeness of the prediction adn the measurement.

3. Resampling:
   Now, a new set of particles are drawn using the weighted probability distribution of the data. Thus, it effectively helps in tracking the robot by localizing its position using the above three steps. The evaluation function of the weight is based on colour and distance between the points in the point cloud.

The drawback of it is that it is exponential in state dimensions which effects the real time tracking of Nao robot so we need to improve the computation speed. This is achieved by using KLD (Kullback Leibler Distance) method for sampling the data.

2.2.2 KLD sampling

This method discussed in [4] increases the speed and reduces the computation time by adapting the size of sample set during the sensor update part of tracking. It is based on the concept that we should use small sample size of particles if the motion of the robot is little and a larger sample size for a larger change in position of the robot. The probability density is shown by the set of samples used. Also, the better the accuracy of the sensor such that it creates less noise, then we require a lower sample set.
This technique adapts the number of samples in particle filter or Monte Carlo methods based on the KL distance ϵ between maximum likelihood estimate $p(x)$ and true probability distribution $q(X)$.

$$\mathbf{KL-distance(p,q)} = \sum \mathbf{p(x)} \log(\frac{\mathbf{p(x)}}{\mathbf{q(x)}}) \qquad (1)$$

The above distance is not a metric because it is not symmetric and does not obey triangle property but it is still widely used due to its applications. The true probability distribution is the data obtained through kinect sensor while the maximum likelihood is the data obtained thorugh prediction of motion dynamics. The required number of samples is inversely proportional to the KL distance obtained.
Thus, it effectively decreases the computation time of particle filter which is exponential in state dimensions.

2.3 Navigation Control

The method of visual servoing provides a closed loop feedback for correcting the errors in odometry which

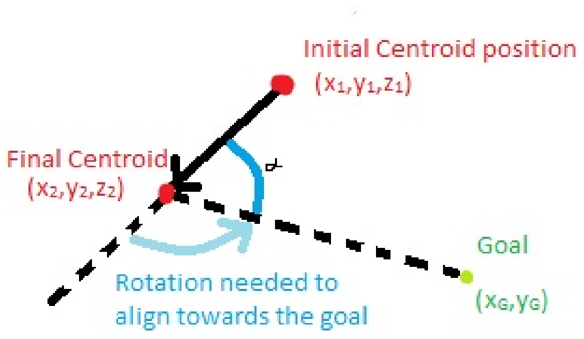


Figure 3: The Small Distance Heuristic : The angle α it needs to rotate.

makes the robot move to the goal position accurately.
We track the centroid of the detected point cloud of Nao as it moves, and estimate its orientation (where Nao is looking) with respect to Kinect using a 'small distance heuristic' described below.
Consider an unknown orientation of Nao. If we give it a small displacement (typically one step) in forward direction (forward, with respect to itself), there will be some change in the tracked point cloud and hence there will be some change in the centroid of the cloud. For small displacements, we assume that the orientation will not change, i.e., the vector connecting the initial centroid position to the final centroid position will be the direction Nao is headed towards.

Once we know the orientation of Nao, we can rotate it towards the goal and move it some significant distance towards the goal. Due to incorrent foot odometry and mechanical errors, it will once again deviate from its path. Therefore, we keep following the 'small distance heuristic' till Nao has reached very close to the goal.

Let (x_1, y_1) be the initial centroid coordinates, (x_2, y_2) be the final final centroid coordinates and (x_G, y_G) be the goal coordinates then the angle α which Nao needs to rotate to align itself towards the goal can be calculated using simple geometry as follows:

$$\mathbf{V_1} = \frac{y_2 - y_1}{x_2 - x_1} \tag{2}$$

$$\mathbf{V_2} = \frac{y_g - y_2}{x_g - x_2} \tag{3}$$

$$\alpha = angleBetween(V_1, V_2) \tag{4}$$

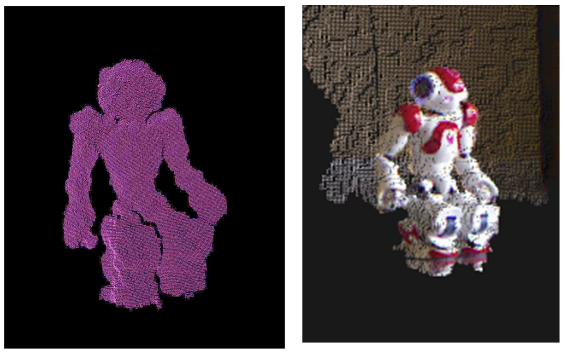
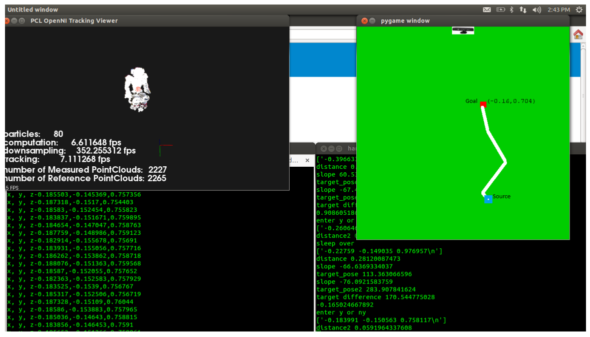


Figure 4: Cluster extracted through the Euclidean segmentation.



```
particles:     80
computation:     6.611648 fps
downsampling:     352.255312 fps
tracking:     7.111268 fps
number of Measured PointClouds:  2227
number of Reference PointClouds: 2265
5 FPS
x, y, z-0.185503,-0.145369,0.757356
x, y, z-0.187318,-0.1517,0.754403
x, y, z-0.18583,-0.152454,0.755823
x, y, z-0.183837,-0.151671,0.759895
x, y, z-0.184654,-0.147047,0.758763
x, y, z-0.187759,-0.148986,0.759123
x, y, z-0.182914,-0.155678,0.75691
x, y, z-0.183931,-0.155056,0.757716
x, y, z-0.186262,-0.153862,0.758718
x, y, z-0.188076,-0.151363,0.759568
x, y, z-0.18587,-0.152055,0.757652
x, y, z-0.182363,-0.152583,0.757929
x, y, z-0.183525,-0.1539,0.756767
x, y, z-0.185317,-0.152506,0.756719
x, y, z-0.187328,-0.15109,0.76044
x, y, z-0.18586,-0.153883,0.757965
x, y, z-0.185036,-0.14643,0.758815
x, y, z-0.183856,-0.146453,0.7591
```

```
['-0.39663
distance 0
slope 60.5
target_pose
slope -67.
target_pose
target dif
0.90860518
enter y or
['-0.26064
distance2
sleep over
['-0.22759 -0.149035 0.976957\n']
distance 0.28120087473
slope -66.6369334037
target_pose 113.363066596
slope -76.0921583759
target_pose2 283.907841624
target difference 170.544775028
-0.165024667892
enter y or ny
['-0.183991 -0.150563 0.758117\n']
distance2 0.0591964337608
```

Figure 5: The implemented method : the pygame window is used to give the goal position through mouse. The white lines show the path followed by Nao.

3 Results

A sample cluster of Nao detected through Kinect can be seen in Figure 4.
The method was tested using different goal positions given as an input on a pygame window. The built-in function *moveTo* in Nao can be given an input to move in its forward direction. The small step used to determine orientation in Section 2.3 was kept as 0.08 metres. After rotation towards the goal, Nao was made to move 0.3 metres forwards towards the goal.
The figure 5 shows a run of this method.

3.1 Problems Faced

The initial attempt of the project focused on mounting Kinect on Nao's head and following the approach discussed in [5]. Figures 6 and 7 show this attempt. But Kinect was too heavy for proper navigation of Nao, and resulted in very aberrated motion because of a great shift in the center of gravity of Nao. Another issue with this

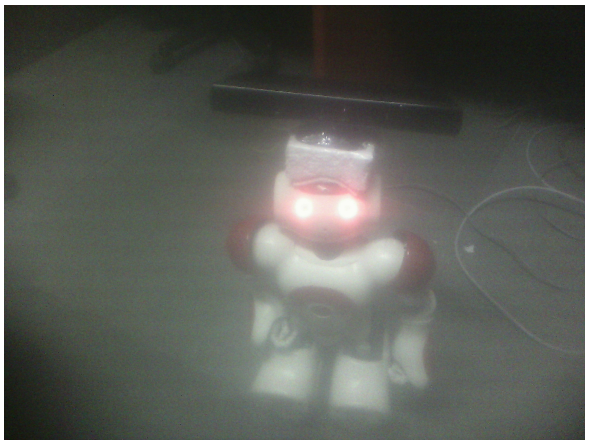


Figure 6: The initial attempt : Mounting on head

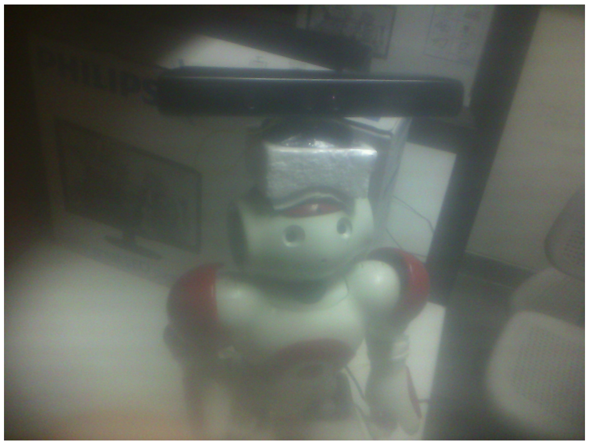


Figure 7: The intial attempt : Mounting on head

approach was that the point cloud generated was quite blurred as the robot was in motion. Thus, the above approach was discarded. Thus, we fixed the kinect in the environment space.
The tracking algorithm could be also used for calculating the orientation of the robot. he issue with it was that the cluster extracted was with a fixed kinect so the points obtained in the cluster of Nao did not completely determine the 3d shape of the robot. So, while tracking the point cloud shape varied by a lot of margin and hence the yaw angle obtained was error prone. Thus, we devised the motion control algorithm which in itself estimated the orientation of the robot.
We faced with another trouble while we were toiling in our project. Nao robot's head got overheated and Naoqi which is the operating system of the robot stopped responding. This created a difficult barrier in our task which was resolved by reflashing the operating system of the robot. A joke was that we had to do open head surgery of Nao for reviving it.

4 Future Work

Rather than giving the goal explicitly on the screen, a goal object can be detected and Nao can be given the goal to move towards the centroid of that cluster. This can further be extended to speech controlled home assisting Nao.
The pose/orientation of the robot can be directly determined by the present state of the cluster extracted through aligning two point clouds. However, doing this in real time is challenging.
Obstacle detection can be employed and obstacles can be detected using the extraction of clusters from the Kinect sensor. Obstacle avoidance algorithms can be also then be employed for finding optimal paths.
The minimum sixteen frames per second for efficient tracking of the robot could not be achieved till now and possible solution for it is to use GPU parallel processing.

References

[1] Creative Commons Attribution 3.0. Euclidean cluster extraction, 2014. [Online; accessed 21-April-2014].

[2] Enric Cervera, Amine Abou Moughlbay, and Philippe Martinet. Localization and navigation of an assistive humanoid robot in a smart environment.

[3] Angela Faragasso, Giuseppe Oriolo, Antonio Paolillo, and Marilena Vendittelli. Vision-based corridor navigation for humanoid robots. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 3190–3195. IEEE, 2013.

[4] Dieter Fox. Adapting the sample size in particle filters through kld-sampling. *The international Journal of robotics research*, 22(12):985–1003, 2003.

[5] Daniel Maier, Christian Lutz, and Maren Bennewitz. Integrated perception, mapping, and footstep planning for humanoid navigation among 3d obstacles. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 2658–2664. IEEE, 2013.

[6] Sebastian Thrun, Dieter Fox, Wolfram Burgard, and Frank Dellaert. Robust monte carlo localization for mobile robots. *Artificial intelligence*, 128(1):99–141, 2001.

[7] R. Ueda. Tracking 3d objects with point cloud library, 2014. [Online; accessed 21-April-2014].