Social Robotics: Human Aware Navigation for NAO

Astha Gupta, Vignesh Durai, Francesco Bruno

Abstract: Navigation is one of the basic skills for autonomous robots. Navigation becomes especially difficult when considering social environment. In a social environment the robots need to consider human comfort and as well as social rules for navigation. This project aims to equip NAO robot with Human aware navigation capabilities. NAO is a small humanoid robot developed by SoftBank Robotics, a very popular and versatile humanoid robot with 25 degrees of freedom including prehensile hands with touch-sensitive sensors, as well as a number of sensors which enables it to perform safe autonomous motion in the environment. In this project we analyse two state-of-the-art algorithms and integrate them with NAO robot using NAOqi Python SDK on ROS.

1. Introduction

Social behaviour such as respecting personal spaces, avoiding collision with another person by changing the speed or local changes to trajectory, avoid blocking another person's path are few examples of unspoken rules that humans employ for their navigation. Thus for a genuine autonomous behaviour the robot needs to take into account the social constraints in addition to basic navigation in dynamic environment.

Autonomous navigation is an important task for. The robot should be able to walk and reach desired position in environments with awareness for human presence. Notably it should take into account human comfort and navigate in socially acceptable manner. Notably, the robot should be able of walking in crowded environments in a natural way, being aware of the surrounding persons, and moving in a socially acceptable fashion.

2. Task Description

The assignment has been divided in three parts:

1. Analysing two approaches given by Kollmitz et. al. [2] and Khambhaita et. al [1]
2. Understanding Naoqi Python SDK 2.8 for programming NAO robot.
3. Integrating the best suited algorithm with NAO robot using ROS framework.

The details of each part is given in the below subsection

3. Analysing approaches for Navigation

The below sections summarise and compare the approach used in the two papers. Section 3.1 recapitulates the approach by [2] and section 3.2 summarises [1]

3.1. Time dependent planning on a layered social cost map for human-aware robot navigation

The approach aims to improve human comfort during navigation by satisfying social space preference. The robot is intended to show polite and obedient behaviour by prioritising human comfort. An home environment has been assumed for their approach. Moreover the focus is on differential drive motion as they urge that holomonic motion might seem to be unnatural and goal oriented for humans.

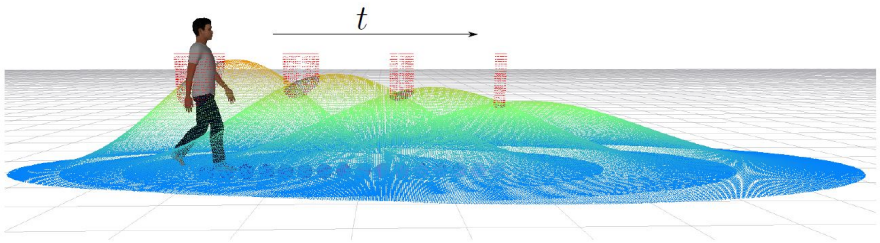


Figure 1: Gaussian-based social cost overlayed with time dependent decay functions for social cost model.

The approach involves developing a social cost model(1) which can be used to generate social costs at different time steps. The snapshot of social cost at different time steps are layered on top of static environment cost to build a dynamic costmap (figure 2). Additionally, a discrete-time graph for all possible configuration of robot at different time steps is generated with help of finite set of executable action (at a given configuration). Finally, A* search algorithm is employed to find the optimize the cost along the global path under social constraints. The optimization is performed considering the dynamic costs which incorporates social and static environment costs, path length and execution time. In order to ensure a constant planning frequency the approach uses a planning timeout to limit the time for planning.

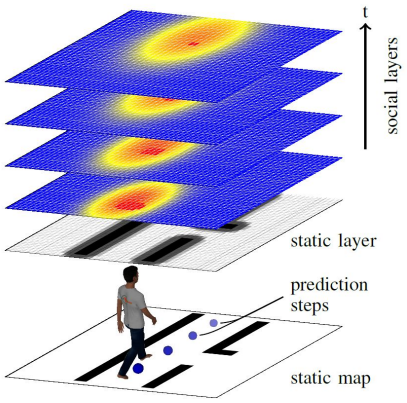


Figure 2: Costmap used for trajectory planning

3.2. Viewing robot navigation in human environment as a cooperative activity

In this paper the aim is to approach navigation as a cooperative activity. In this approach the cooperative planner generates a trajectory for robot as well as for human (which they may or may not follow). The cooperative situations have been simulated by providing trajectories planned for humans from the cooperative planner to human navigation simulator.

The approach is to extend timed-elastic bands technique to formalise optimization problem as a non-linear least square problem. The approach incorporates the prediction and optimization of human trajectories. Thus the approach tries to jointly optimise the trajectories of robot and human for social and kinodynamic constraint.

The trajectory is represented by a tuple, $\mathcal{B} := \{\mathcal{P}, \mathcal{T}\}$, where $\mathcal{P} := \left\{p_i = [x_i, y_i, \theta_i]^T\right\}_{i=0}^{n}$ is the position and

$\mathcal{T} := \{\Delta t_j\}_{j=0}^{n-1}$ is time. The $\mathcal{B}_\mathcal{R}$ denotes the trajectory of the robot and $\mathcal{B}_{H_k}$ is the trajectory of the k^{th} human.

$$f\left(\mathcal{B}_\mathcal{R}, \mathcal{B}_{H_k}\right) = \sum_a \gamma_a f_a\left(\mathcal{B}_\mathcal{R}\right) + \sum_b \gamma_b f_b\left(\mathcal{B}_{\mathcal{H}_k}\right) + \sum_c \gamma_c f_\mathcal{C}\left(\mathcal{B}_\mathcal{R}, \mathcal{B}_{\mathcal{H}_k}\right) \tag{1}$$

$f_\mathcal{A}$: Cost function for kinodynamic and nonholonomic constraints on Robot.

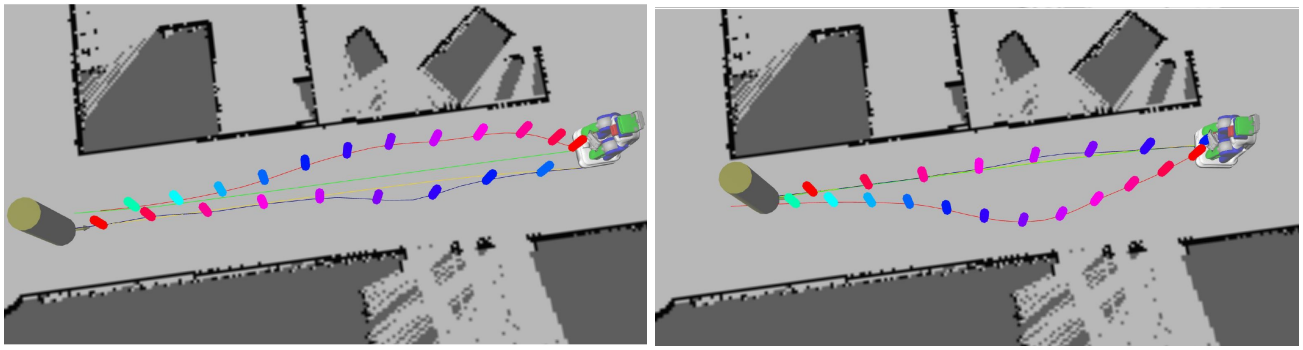
$f_\mathcal{B}$: Cost function for kinodynamic constraints on Humans.

$f_\mathcal{C}$: Cost function for Human Robot social constraints.

Thus the optimal trajectories are obtained by minimizing the cost function as below.

$$\left\{\mathcal{B}^*_\mathcal{R}, \mathcal{B}^*_{\vartheta_k}\right\} = \underset{\{\mathcal{B}_\mathcal{R}, \mathcal{B}_{\mathcal{Y}_k}\}}{\arg\min}\, f\left(\mathcal{B}_\mathcal{R}, \mathcal{B}_{\mathcal{Y}_{f_k}}\right) \tag{2}$$

The approach uses g^2o framework which utilises the Levenberg-Marquardt method for solving non-linear least squares problems defined by the graph structure.



(a) (b)

Figure 3: A corridor-crossing scenario for Human(green path) and Robot(red path). Part (a) shows the proposed cooperative plan. Part (b) shows the plan if human rejects the solution. The small cylinders on top of the trajectories shows future positions at every second. The color of the cylinders correspond to same future time.

Figure. 3 illustrates the working of the algorithm. The cooperative planner calculates and proposes trajectory for robot and human respectively. The robot follows its trajectory proactively(without waiting for human), moving to the right, as a way to offer a solution for co-navigation to human. If the robot rejects the prroposed solution (by not moving along the provided solution), the robot adapts to the human's movement.

3.3. Comparison

Although, the second approach given by [1] implements a cooperative behaviour which is a more inclined to be socially acceptable and is commonly observed in human navigation, the approach lacks an evaluation against a benchmark in comparison to the first one ([2]). The first approach compares their algorithm against static planning and reports test for multiple scenarios.

Moreover, the first approach provides a framework for simulating and detecting humans, as apart of their open-source contribution(human aware navigation) along with a wiki on how to use the package. While the second approach only provides the source code for local planner(hateb local planner). Therefore, we choose the first approach for integration with NAO robot

4. NAOqi 2.8 Python SDK and ROS

NAOqi provides APIs as part of their Python SDK for ease of programming. The NAO robot available in lab is version 6, which works on NAOqi 2.8 and requires the latest SDK specifically developed for version 6 NAO. For the project we used the following APIs:

- ALMotion: For moving the robot by giving it a velocity or position and getting the odometry of the robot.
- ALRobotPosture: For commanding the robot to go in different postures before and after motion commands.

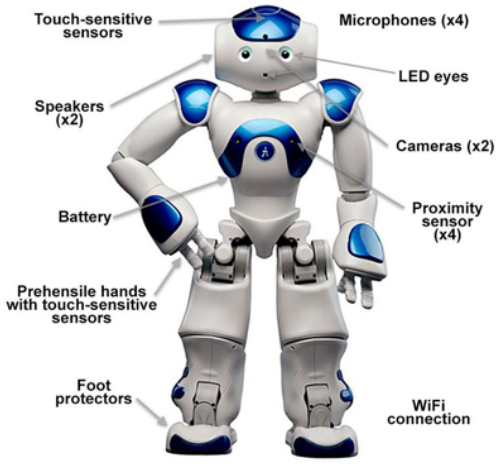


Figure 4: NAO Robot

Although, there are a lot of packages and support for NAO on ROS. The APIs used for integration of NAO with ROS are based on previous versions of NAOqi. Which lead to unexpected problem while using the NAO packages on ROS. For example, one of the script called "nao_walker.py" provided by "nao_apps" is currently using deprecated APIs which lead to problems such as, when zero velocity is provided to the robot through "cmd_vel" topic, the robot falls down.

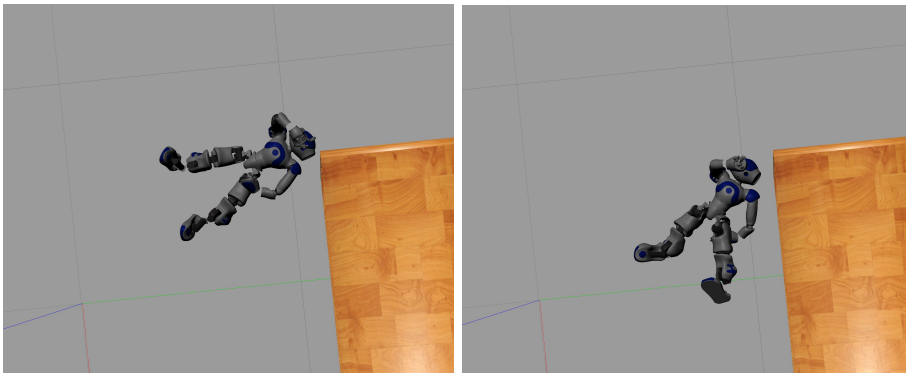
Therefore, we developed a new script using the same template as the original script but with new APIs suitable for NAO v6. Please find the script on the github page of this project.

5. NAO Integration for Human Aware Navigation

For integration of NAO with Kollmitz et. al.[2]. We employed two different methods. Using simulation and using Real NAO robot. The following section details the challenges for simulation and implementation details for Real NAO robot.

5.1. NAO Simulation

ROS provides an easy way to simulate NAO robot in gazebo using URDF. However using a simulation posed a big challenge as the controller for simulation are not well implemented and thus robot falls down once the simulation is started. Due to this difficulty we decided to circumvent this problem by directly using the real NAO robot.



(a) (b)

Figure 5: NAO Virtual gazebo simulation

5.2. NAO Robot

The NAO robot is integrated for testing with help of the new "nao_walker.py" script developed for this project(as mentioned in section 4). Apart from this script, there were few more challenges. These challenges and our approach to tackle them are detailed below.

1. NAOqi uses Normalised speed: The NAOqi API uses normalised(between -1 and 1) speed as input. But the algorithm ([2]) provides an absolute velocity for the robot to follow.

2. The algorithm for Human aware navigation requires laser data and odom from the Turtlebot to get the position of the robot in the world frame and provide velocity commands : Thus we can not simply replace the odom of turtel bot with the odometry of NAO (using getRobotPosition) as laser depends on the odometry of the Turtlebot. Thus if instead of using odometry of Turtlebot if we substitute it with odometry of NAO, the laser data will not be correct.

Thus we tried couple of potential fixes:

- *Provide same velocities to Turtlebot and NAO*: The mapping between absolute velocity of NAO and the normalised input that can be provided to the robot is not linear. Also the absolute velocity in different directions for the same input (for example 0.1 in linear-x linear-y and angular-z) is different. Thus, a direct mapping of the command velocities generated by planner to NAO does not provide movements in the expected directions

- *Scaling down the velocity of Turtlebot*: We tried to scale down the velocity provided by the planner for Turtlebot to match the order(10^{-2}) of NAO's absolute velocity. This causes the Turtlebot to move very slowly. Therefore, the planner need to keep updating the path. While the planner is updating the path, the velocities suggested by the planner are null. Thus, the NAO never gains sufficient velocity to move away from the initial position as it is provided with zero velocity commands after every few seconds.

- *Using a large map for simulation*: We assumed that a large map will help us scale up the distance travelled by NAO robot in real world, in order to make some meaningful observation for the path taken by the robot. But, since the time has a threshold on planning time. Using a goal position very far away from the initial causes the planner to time-out before finding a feasible path.

- *Reducing the min and max velocity of the planner*: Reducing the min and max velocities of the planner to be comparable to the absolute velocity of NAO robot causes planning time-out. As the possible movements in a given time step becomes very small and thus the algorithm is not able to find a suitable path in the given time threshold.

- *Increasing planning time and decreasing planning frequency*: For the cases where a path was not found due to time-out, we tried to increase the planning time and decreasing the planning frequency. But the problem still persisted.

- *Using the full range of NAO velocity*: We tried to provide planner with min and max velocity as -1 to 1(NAO's range of velocities), and use the absolute velocities from NAO's API (getRobotVelocity) to the Turtlebot but this didn't produce the desired movement as well.

- *Using odometry and fake laser data for NAO*: In an attempt to avoid using Turtelbot simulation. We created "fake_laser" for NAO robot. Assumption here is that static map contains all the information of obstacles in a given environment and thus the costmaps will help avoid collision with these obstacles. We used the TF tree and Odometry from naoqi_driver[1] along with the fake laser (attached to base_link of robot). But since the generated Planner velocities by move_base are absolute, and NAO's APIs assume normalised velocities. The robot was not able to achieve the desired pose.

- *Using NAOqi's "move" function*: In order to improve upon the previous limitation, we used NAOqi's move function instead of "moveToward". For this approach we changed the min and max velocities parameters of the planner to ensure that the robot is provided with a reasonable range of velocities.

[1]Note: only naoqi_driver should be used instead of nao_bringup, as the later might give unexpected behaviour due to old APIs

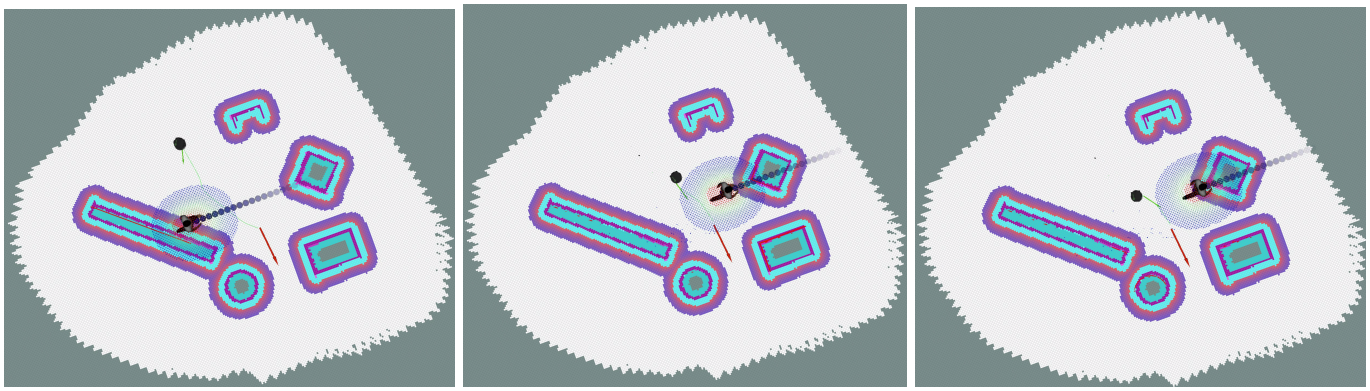
```
min_vel_x: 0.0
max_vel_x: 0.04
acceleration_x: 0.04
min_vel_phi: -0.30
max_vel_phi: 0.30
acceleration_phi: 0.1
# planner
planner_frequency: 1.0
planning_timeout: 0.90
```

But the results were not satisfactory.

6. Result and Discussion

In the preliminary tests, with the original implementation given in [2], the robot is successfully able to re-plan its trajectory or wait for the human to pass while respecting their comfort zone. This can be clearly observed in Fig. 6.

In part (a) the robot plans an initial trajectory passing through the comfort zone of human, given by the Gaussian distribution. When the robot is near the human, in part (b), the robot waits for the human to pass, while re-planning its path in order to avoid the social cost as shown in part (c).



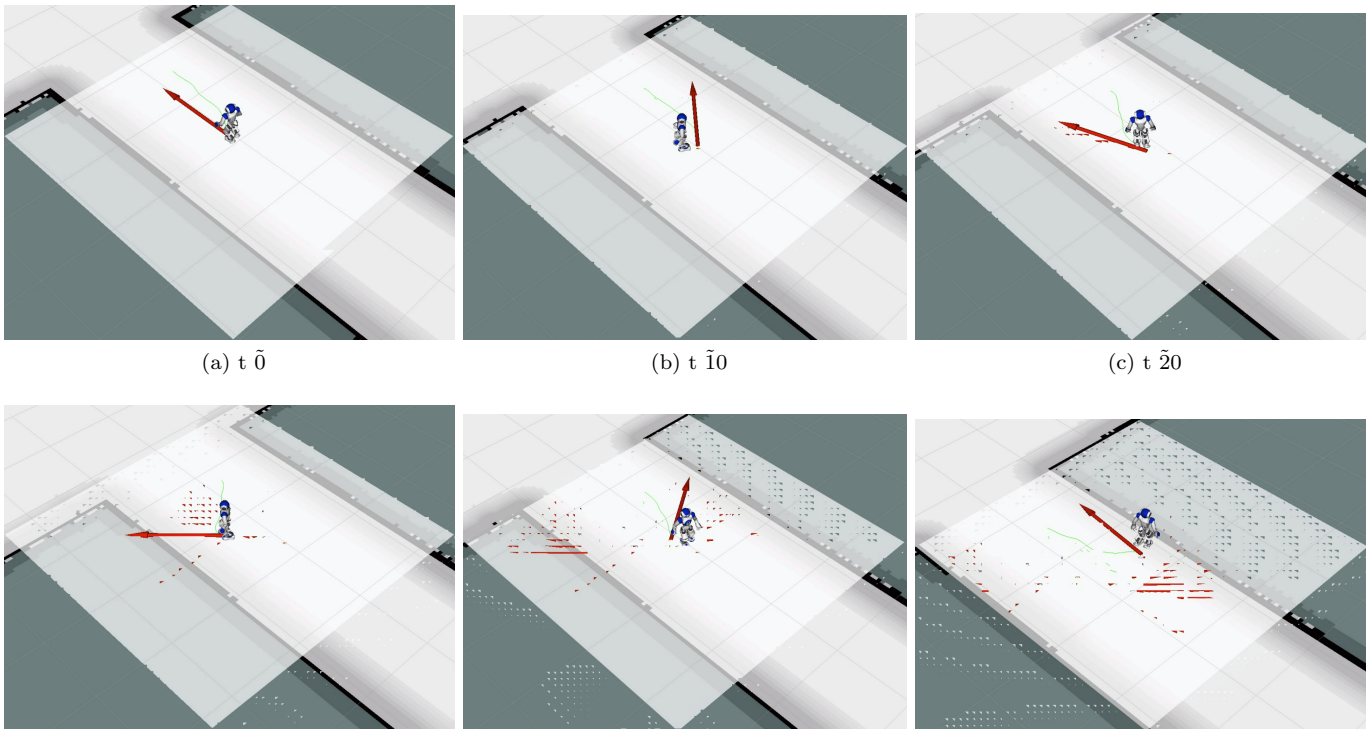
(a) (b) (c)

Figure 6: .

At this point in the project the, we have implemented a new "walker" script which uses the latest NAOqi APIs to connect to the robot and provide commands for motion. All the required TF transformations has been implemented for visualisation of NAO robot's motion(not the simulation) on rviz on any map file. Note that for NAO the local and global tolerance for TF should be set to a higher threshold(example 1.5 2.5 sec). This is due to difference in synchronisation between the Robot and compute engine(laptop/desktop).

Figure. 7 illustrates on of the experiments for NAO which uses its Odometry and fake_laser along with "move" function from NAOqi API. As we can see that with the right parameter tuning for the planner, a possible trajectory is generated. But, given the command velocities from the planner, NAO is unable to correct orient itself. Therefore, it is unable to follow its trajectory.

The red arrow in Figure. 7 represents the odometry (2D Pose). Thus, it can be clearly seen that the robot's actual pose and the odometry data are not synchronised correctly. A possible explanation for this behaviour can be the time-difference between different TF nodes, as mentioned earlier.



(a) t $\tilde{0}$ (b) t $\tilde{1}0$ (c) t $\tilde{2}0$

(d) t $\tilde{3}0$ (e) t $\tilde{4}0$ (f) t $\tilde{5}0$

Figure 7: The green line is the path that robot needs to follow to reach the desired goal position. Red arrow is the optometry marker for the robot. The images show various pose of the robot at different time instant for one experiment.

7. Future Work

Next possible steps for fixing the behaviour of NAO's navigation is to tune the parameters of various cost-maps (local, global and lattice). These parameters should be tuned carefully to avoid any damage to the robot. Apart from parameters, the robot's behaviour can be improved by potentially reducing the time gap for TF tree and therefore trying a reduced tolerance for cost-map.

Other possible directions includes implementing the approach given by [1] using the "people_detection" package provided by [2] and tuning the navigation parameters to allow navigation with slow velocities. As seen in this project, the navigation algorithms are generally designed for mobile robots. Since mobile robots are generally faster than humanoids, there are many caveats in using algorithms designed for mobile robots with humanoids. Therefore, in order to equip humanoids with socially aware navigation, it is essential to develop algorithms keeping their limitations in mind.

References

[1] Harmish Khambhaita and Rachid Alami. "Viewing robot navigation in human environment as a cooperative activity". In: *Robotics Research*. Springer, 2020, pp. 285–300.

[2] Marina Kollmitz et al. "Time dependent planning on a layered social cost map for human-aware robot navigation". In: *2015 European Conference on Mobile Robots (ECMR)*. IEEE. 2015, pp. 1–6.