Localization of persons in a room

Group Members:
Shiva shankar ARUMUGAM
Varsha DEVI
Kajal PURI
Koutaiba CHAKER

Date: 29/04/2019

Outline

- Objectives
- Introduction
- Architecture
- Detector Node
- Tracker Node
- Conclusion

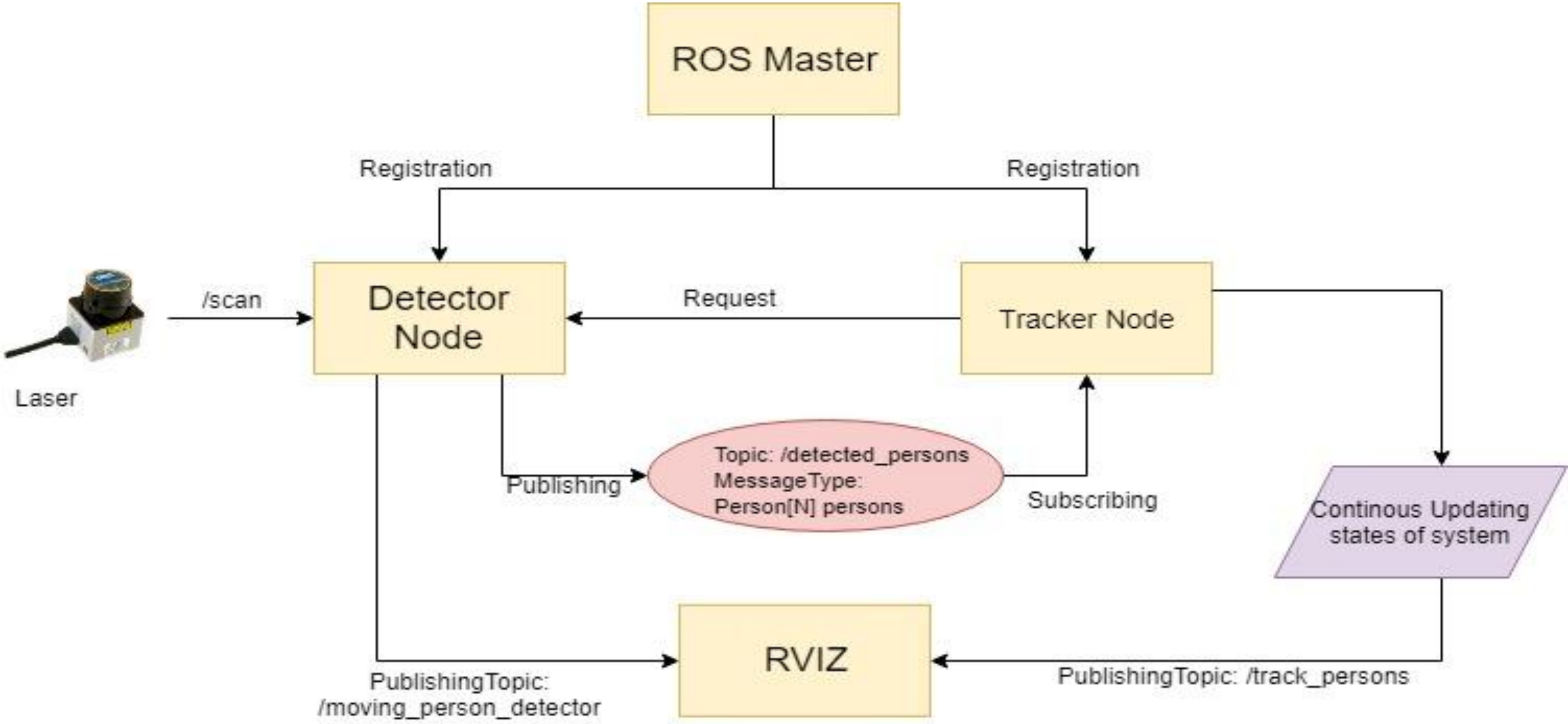
Objectives

- To study different steps from sensor perception to action
- Study Multi-Person detection & Tracking
- Create Detector Algorithm
- Create Multi Person Tracking algorithm

Introduction

- A static LaserScanner installed in a room
- Localization of moving persons in a room
- Two Nodes communicating with each other
  - Detector
  - Tracker

Architecture



Detection

Clustering hits of the laser

- We need to cluster hits to form objects
- close hits belong to the same object
- Laser data are acquired in the trigonometric way

Clustering Algorithm

- Initialization of the first cluster
- Create a first cluster with the first hit/beam
- For all the beams except the first one
  - If the absolute distance between the current hit/beam and the previous hit/beam is lower than a given threshold Then add the current hit/beam to the current cluster
  - Else create a new cluster with the current beam

  End for

Detection

- After Clustering, detection of moving legs
- Steps:

```
if
    the size of the current cluster is higher than "leg_size_min" and
    lower than "leg_size_max" and
    it is greater or equal "dynamic_threshold"% of its hits that are dynamic
then
    the current cluster is a moving leg
if
    the distance between two moving legs is lower than "legs_distance_max"
then
    we find a moving person
```

- At the end the detected persons are published which was stored in PersonArray Message
  - to the tracker node
  - to Marker for visualization on RVIZ

Tracking

- PersonArray[] from Detector node is received and converted into an array of DetectedPersons object inside Tracker node
- TrackedPersons[] - obj attributes include color -> random RGB values for each new track, incremental id_num, kalman filter related constants and matrices

Algorithm - Nearest neighbour matching

```
foreach detected_person in detected_persons:
    foreach tracked_person in tracked_persons:
        dist = get_distance(detected_person_pos, tracked_person_pos)
        if dist < threshold:
            update_dist(detected_pos, tracked_pos, dist)
foreach potential_tracks, existing_tracks:
    if dist_between(potential_track, existing_track) < threshold:
        Matched_tracks[existing_track_idx] = potential_track
```

Kalman filters

- Kalman filters are linear models for state estimation and follows the "observe, predict, update" paradigm
- pykalman is a Python library that provides a simple interface to use KalmanFilter just by providing the transition and observation matrix along with its corresponding covariance matrices
- Linear equation of constant velocity motion model of the system

$$\mathbf{x}_{k+1} = A_k \mathbf{x}_k + B_k a_k + r_k^{(s)}$$

$$\mathbf{x}^{(P)} = \underbrace{\begin{pmatrix} 1 & 0 & t & 0 \\ 0 & 1 & 0 & t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{A_k} \mathbf{x}_k + \underbrace{\begin{pmatrix} 0.5t^2 & 0 \\ 0 & 0.5t^2 \\ t & 0 \\ 0 & t \end{pmatrix}}_{B_k} \cdot a_k$$

r_k - Gaussian noise in the system (initially approx by std_dev and variance)

KalmanFilter's prediction step uses this noise to calculate covariance and based on which it updates the next state by filtering out noise

Conclusion

- Our tracking algorithm entirely depends upon the accuracy of detector node so detector needs improvement
- KalmanFilter needs to be studied and explored further to better use it in our system to fine tune the tracking accuracy
- Handle occlusions in crowded environments
- Performance improvements - reduce processing time

Thanks for your attention
Questions???