Lidar filters

You have been assigned to write filters to reduce noise in the data coming from a LIDAR sensor attached to your robot. The LIDAR generates scans at a certain rate. Each scan is an array of length *N* of float values representing distance measurements. *N* is typically in a range of ~[200, 1000] measurements, and it is fixed. Measured distances are typically in a range of [0.03, 50] meters. Each time a scan is received, it will be passed on to the filters. Each filter object should have an **update** method, that takes a length-*N* array of ranges and returns a filtered length-*N* array of ranges.

We want you to write two different filter objects:

- A range filter

  The range filter crops all the values that are below range_min (resp. above range_max), and replaces them with the range_min value (resp. range_max)

- A temporal median filter

  The temporal median filter returns the median of the current and the previous D scans:
  $y_i(t) = median(x_i(t), x_i(t-1), \ldots, x_i(t-D))$

  where x and y are input and output length-*N* scans and i ranges from *0* to *N-1*. The number of previous scans *D* is a parameter that should be given when creating a new temporal median filter. Note that, although the **update** method will receive a single scan, the returned array depends on the values of previous scans. Note also that the for the first *D* scans, the filter is expected to return the median of all the scans so far.

  Here is a short example of the result (Y) of a temporal median filter object with *D*=3 for an input (X) of dimension N=5, for the first five updates:

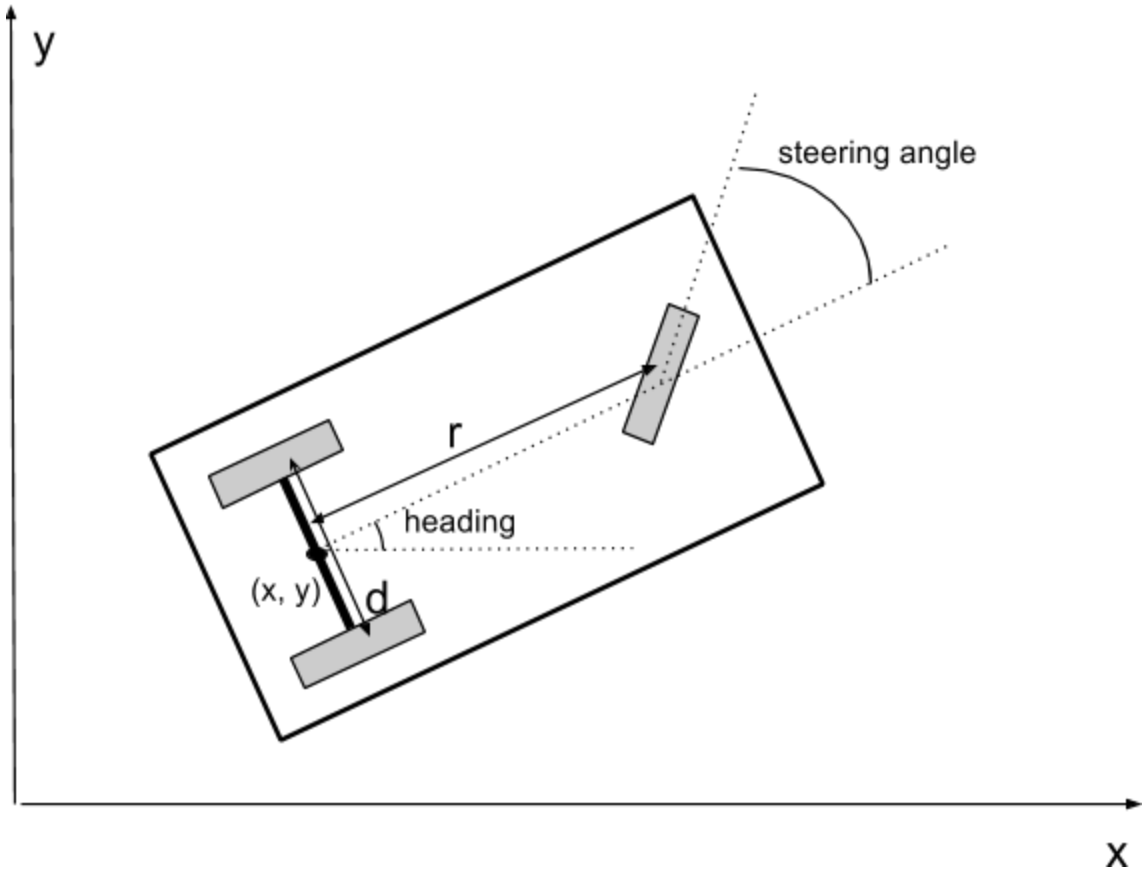
T (time)	X (input scan)	Y (return of the update)
0	[0., 1., 2., 1., 3.]	[0., 1., 2., 1., 3.]
1	[1., 5., 7., 1., 3.]	[0.5, 3. , 4.5, 1. , 3.]
2	[2., 3., 4., 1., 0.]	[1., 3., 4., 1., 3.]
3	[3., 3., 3., 1., 3.]	[1.5, 3. , 3.5, 1. , 3.]
4	[10., 2., 4., 0., 0.]	[2.5, 3. , 4. , 1. , 1.5]

You are expected to write documentation and test correctness for your code.

You can either use Python 2.7 and/or C++. For Python, Numpy library may be used. For C++, boost and stl libraries may be used

Pose estimator

You have been assigned to write a pose estimator designed to estimate the 2D position of the mobile platform (x, y, heading) based on odometry information using wheel encoder and imu data. The mobile platform is a wheeled tricycle with a steering mechanism that has been attached to the front wheel. *r* is a distance from the front wheel to back axis, *d* is a distance between the rear wheels.



We use a right handed coordinate system, where Z is up, X is forward and Y is to the left of the robot, with positive Yaw (around the z axis) being counter-clockwise and negative yaw being clockwise. The front wheel can rotate up to 90 degrees in both directions around the Z axis. The platform is moving on a 2D-plane (constant z=0). The mobile platform is expected to go straight when the steering angle is 0.

Parameters of the tricycle:

Front wheel radius = 0.2 m
Back wheels radius = 0.2 m
Distance from front wheel to back axis (*r*) = 1m

Distance between rear wheels (*d*) = 0.75m

A traction motor has been attached to the front wheel and is equipped with an encoder with a resolution of 512 ticks per revolution. The steering mechanism also includes an absolute encoder which provides an estimation of the steering angle (in radians).

The pose of the platform is a pose of the center of the rear axis. Initial pose is assumed to be (x, y, heading) = (0, 0, 0).

At each step, the new estimate of the position of the mobile platform is obtained through a call to an *estimate* method with the following API:

estimate (time, steering_angle, encoder_ticks, angular_velocity) *returns* estimated_pose (x, y, heading)

Input	**Description**	**Unit**
time	Time of reading of the input data	sec
steering_angle	Steering wheel angle	rad
encoder_ticks	Number of ticks from the traction motor encoder	Ticks (integer)
angular_velocity	Reading from a gyroscope measuring the rotation velocity of the platform around the Z axis	Rad / s

Output	**Description**	**Unit**
estimated_pose	New estimated pose. Tuple (x, y, heading) representing the estimated pose of the platform.	(m, m, rad)

You are expected to write documentation and test correctness for your code.

You can either use Python 2.7 and/or C++. For Python, the Numpy library may be used. For C++, stl, boost libraries may be used.