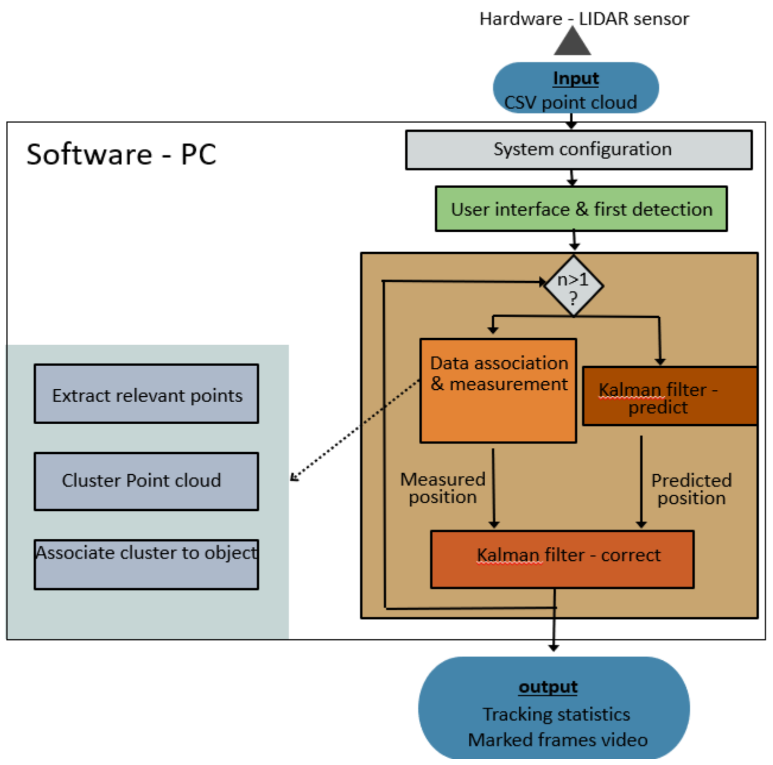
LidarTrackerV1 user’s manual

By Dan Komisarchick & Ofek Weiss

1.Introduction

This project was conducted as part of a research which was carried out at the TAU Garden for Zoological Research. The scientific study of small flying creatures poses a significant challenge when it comes to monitoring their movement. To address this challenge, the project introduces a software system, using MATLAB, specifically designed to identify and track small flying objects.

Figure 1: Block diagram



This document mainly will focus on the software implementations, configurations and use.

It will provide an overview about the system’s structure and functionality and will provide a basic manual to operate and use the system.

For extra information please refer to the github repository:

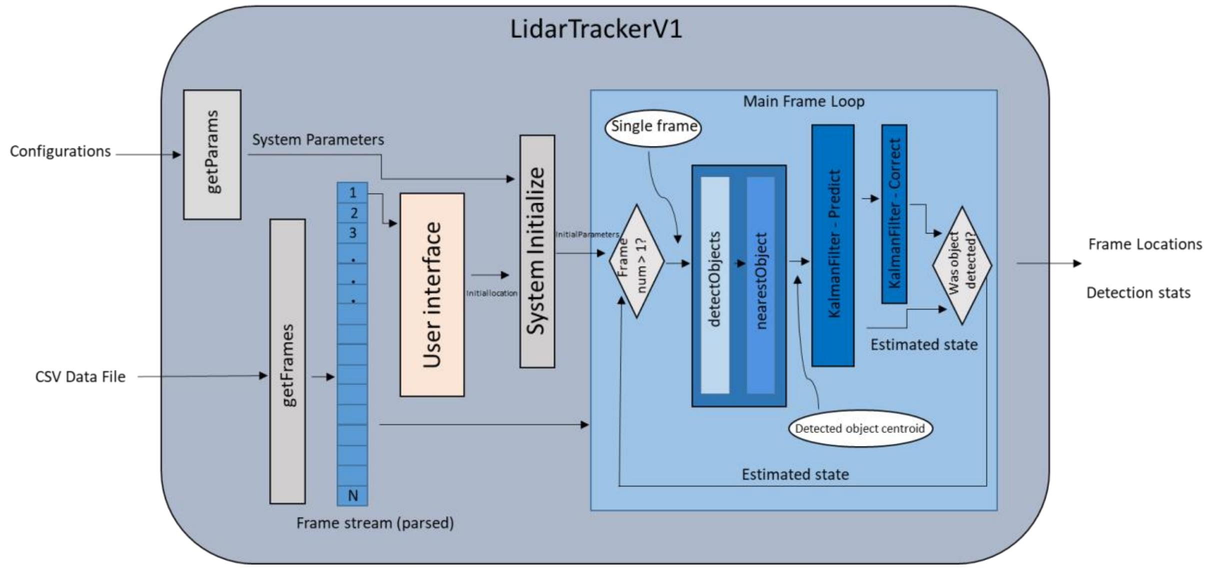
https://github.com/Dankomis/FinalProject2023

That will include the source files and extra documentation.

2.Software implementation

In this section the system's submodules implementation and functionality will be described.

Figure 2: System implementation block diagram



The figure above visualizes the main submodules of the system and the workflow of the software that implements it.

Each module will be described in the following paragraphs.

2.2 System parameter acquisition

Described as getParameters. This module gets a data structure that holds the configuration information of the system as an input and outputs all the relevant system parameters for initialization and execution of the system.

The motivation for the configuration process is the main understanding of the project that the system should support different cases of tracking for multiple types of objects, environments and different LIDAR sensors. Therefore the system is completely configurable, for example the time of each frame, number of points in each frame and the type of Kalman filter that will be used based on the motion model.

Frame parsing

Based on the LIDAR sensor that was used, this module will parse the raw data stream into frames according to the configured frame duration and number of points. The data will be assigned as n (number of frames) matrices that contain x,y,z and reflectivity columns for each point (row) and will be saved into a data structure.

This preprocess module will allow us to obtain the data in a rather simple structure and will save complexity and space.

2.3 User interface

This module will output the initial location of the object that will be tracked and researched. By implementing the function "*initPos()*" the user will be prompted to mark with MATLAB data tip[1] the centroid of the object, this mark will be used as the initial location to initialize the tracking system.

The idea of using this user interface came after a discussion with the zoological research team. As the users of the system they have mentioned the variety of subject's they are studying and complexity of modeling the objects for detection, and on the other hand the need in a system that will help to learn the motion models of the objects. Therefore the key assumption is that the researchers can identify the objects and are focused on researching the movement of different flying subjects this module was implemented.

2.4 Main frame loop

The initial frame will help to initialize the frame loop and this module will iterate each frame to estimate the object's state vector.

The first submodule "*getMeasurement"* includes two main key features:

- Object segmentation - using the estimated location from the previous frame a "gate" will be created, this is a region of interest (ROI) configured according to the object's estimated speed. In the ROI all objects will be segmented into clusters by using a cluster algorithm. Different algorithms were compared and the from the results we concluded that pcsegdist[2] will be the choice.

  Note: this module and segmentation algorithm is configurable as well.
- Data association - using the concept of minimum Euclidean distance[3] the centroid that represents the object will be chosen.

[1] MATLAB data tip - https://www.mathworks.com/help/matlab/ref/matlab.graphics.datatip.datatiptemplate-properties.html?s_tid=srchtitle_data%20tip_2

[2] Pcsegdist - https://www.mathworks.com/help/vision/ref/pcsegdist.html#mw_c75a2c24-9ee9-4d38-b9e6-f974ffc7bc58_seealso

[3] See theoretical background section 2.4

This submodule outputs two arguments, the measured location and a flag which indicates if the object was detected.

The functionality of the submodule is a measurement function that outputs an observation that will be used to correct the estimation of the Kalman filter.

The second module includes the Kalman filter it uses two steps:

- Prediction - using the configured motion model and noise parameters, the current state is estimated, based on the previous frame state.
- Correct - the Kalman gain is computed based on the current and previous frame parameters, and the state estimation will be corrected by weighing between the prediction and the measurement computed from the first submodule. If the first submodule has failed to produce a measurement the correction step won't be used.

2.5 system's output

The system is described as a closed function that can be configured by an outer program and will simulate the system with the desired data files and configurations.

Using a data structure "*verif*" that will be filled during runtime with tracking information per frame, the program that simulates the system gets from the system variety of insights: video stream with tracking bounding box, location and velocity statistics, module success rates etc.

3. Parameters and configurations

The system is completely configurable for the user's needs. The user can configure the system according to the type of LIDAR sensor in use, the environment and the motion model of the tracked object.

The system will be configured by the application that simulates the system[4].

Configurations will be executed using:

- *config: struct that represents the system's configuration containing the following fields*

    config.kalman_filter - the type of Kalman filter that will be used

    config.meas - the type of detection function that will be used in each frame

    config.frame_time - frame duration in seconds

    config.frame_points - number of points per frame

    config.ROI_flag - flag that indicates if ROI of room should be chosen

    config.ROI_flag - flag that indicates if ROI of room should be

    config.video_flag - flag that indicates if video stream will be created

This data structure will be further used by *getParams() to initialize the system's parameters.*

[4] Check TrackTester.m, LidarTrackerSimulation.m application files in github repository

4. Using the system

In the following section the use of the system will be described including how to configure, simulate and extract relevant data.

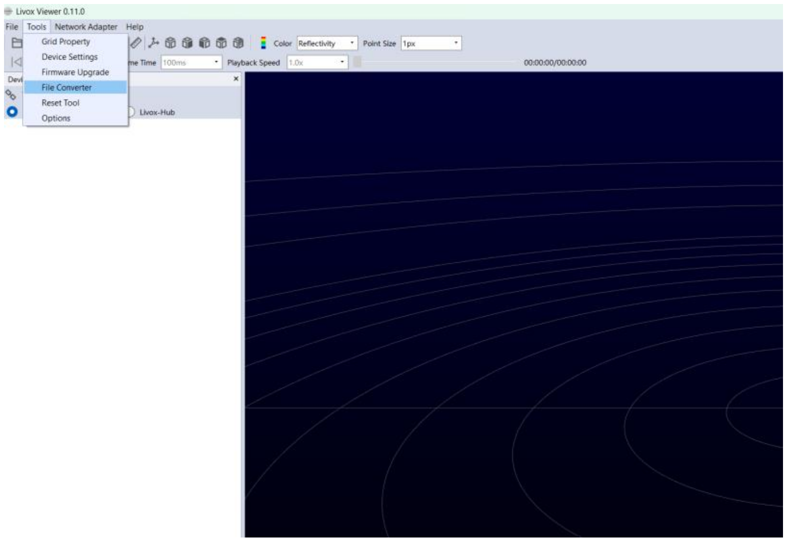
4.1 Data input

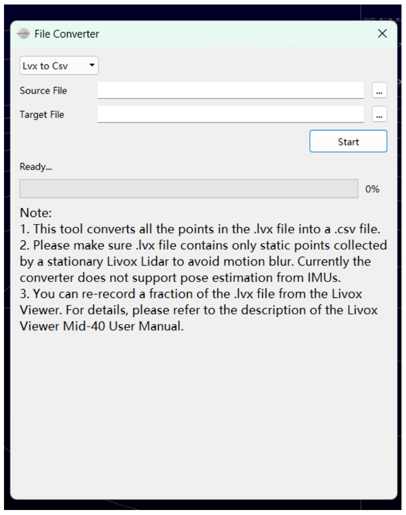
The point cloud stream input will be of a CSV type. After recording the LIDAR data make sure you save the point cloud stream as a CSV file or convert it into a csv file.

Example using Livox platform:

Step 1 - record the stream using LivoxViewer SDK and save your work, it will be saved as an LVX file.

Step 2 - On the LivoxViewer option bar go to: Tools - file converter choose LVX to CSV and apply conversion.





Note:
1. This tool converts all the points in the .lvx file into a .csv file.
2. Please make sure .lvx file contains only static points collected by a stationary Livox Lidar to avoid motion blur. Currently the converter does not support pose estimation from IMUs.
3. You can re-record a fraction of the .lvx file from the Livox Viewer. For details, please refer to the description of the Livox Viewer Mid-40 User Manual.

4.2 Simulation app

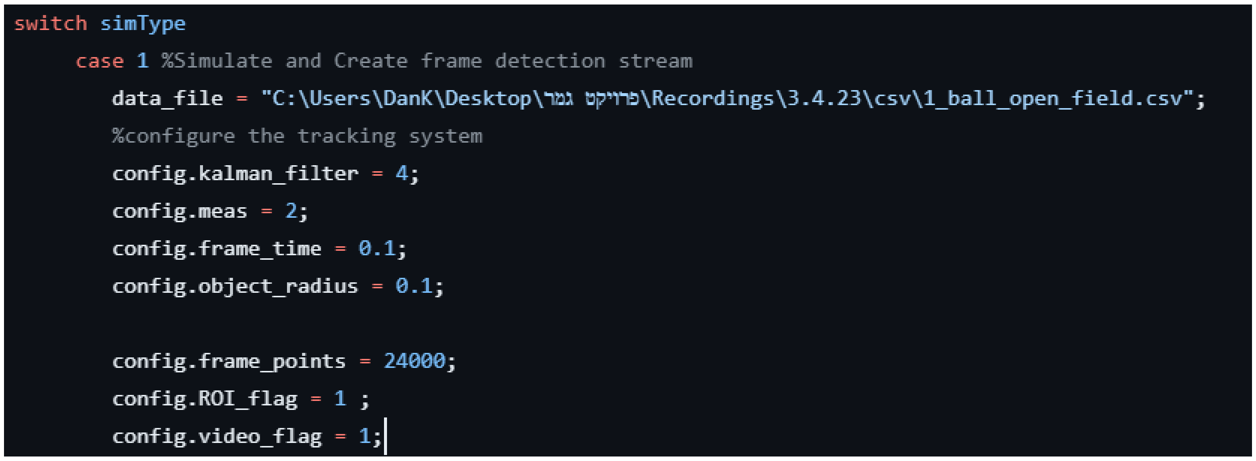
To simulate the system there will be needed an external executable MATLAB™ file to simulate the system and call the function of the tracker.

It is possible to create new simulations or use the existing tests available in the github repository.

There are a few basic steps to simulate the system.

Step 1: Load CSV input data files for the tracking process

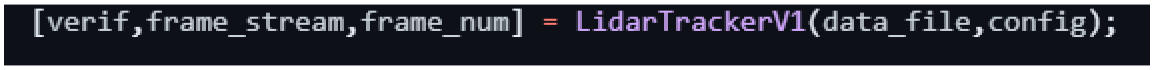
Step 2: For each simulation as was explained in previous sections a configuration is needed



```
switch simType
     case 1 %Simulate and Create frame detection stream
        data_file = "C:\Users\DanK\Desktop\פרוייקט גמר\Recordings\3.4.23\csv\1_ball_open_field.csv";
        %configure the tracking system
        config.kalman_filter = 4;
        config.meas = 2;
        config.frame_time = 0.1;
        config.object_radius = 0.1;

        config.frame_points = 24000;
        config.ROI_flag = 1 ;
        config.video_flag = 1;
```

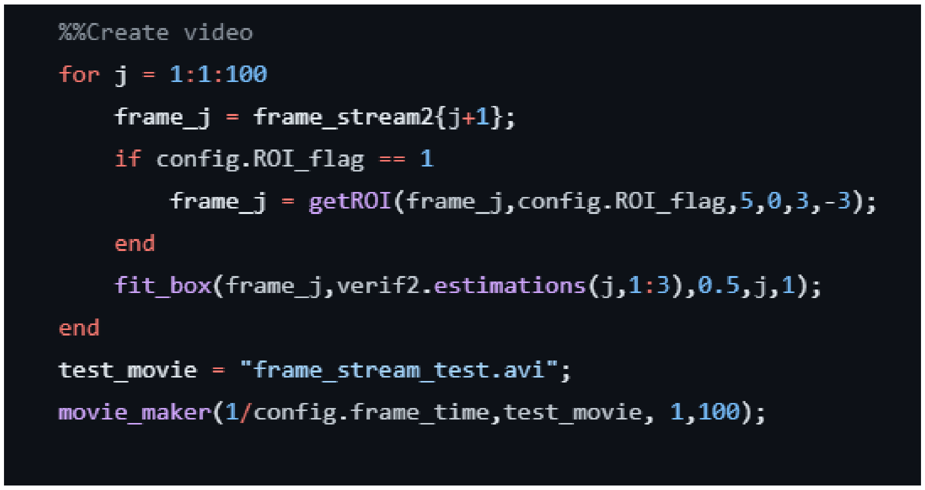
Step 3: Call LidarTrackerV1 with relevant parameters.



```
[verif,frame_stream,frame_num] = LidarTrackerV1(data_file,config);
```

Step 4: This step can be different for each type of simulation. Basically it will be needed to make sure to analyze or process the raw output data to the desired output.

In the example below a video stream of tracking is created:



```
%%Create video
for j = 1:1:100
    frame_j = frame_stream2{j+1};
    if config.ROI_flag == 1
        frame_j = getROI(frame_j,config.ROI_flag,5,0,3,-3);
    end
    fit_box(frame_j,verif2.estimations(j,1:3),0.5,j,1);
end
test_movie = "frame_stream_test.avi";
movie_maker(1/config.frame_time,test_movie, 1,100);
```