Platform Documentation

Introduction

This documentation provides the information required to run and modify SecureLoc project. It does not aim to provide accurate documentation on the platform code, the python documentation being provided elsewhere.
The hardware and software requirements are described, instructions on how to install and where to find the tools needed are given, and the basic usage of the 3D Engine is explained.
Hardware section describes the material you would need to run this project.
Software section provides instructions on the installation of the various libraries required. Ensure that you followed EVERY step if any issue when attempting to run the project.
3D Engine Manual section describes the usage of the 3D engine once operational, and gives insights and how to change some useful parameters when running SecureLoc.

Table des matières

Hardware

Two types of nodes: **Anchors** and **Tags.**
Hardware description :

Anchor :

TeensyDuino 3.2 --- (Serial – USB) -- Raspberry 3B --- (Ethernet) --- Switch)–-(Ethernet) ---Server

Robot :

TeensyDuino 3.2 --- (Serial – USB) --- (Monitor * [Any Laptop or CPU])

Server :

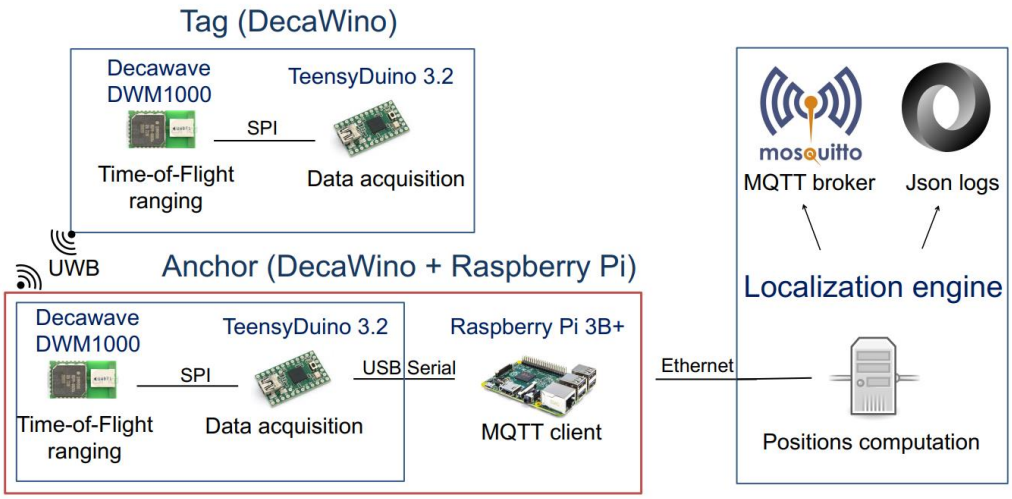
Windows CPU (can run on any OS, Python code)

3D Engine:

Windows CPU ** (can be run on any OS, Python code)

* Not required, only for debugging
** Can be used on the same CPU as the server

Figure 1- Hardware Architecture



Software

Teensyduino:

Development with Arduino IDE. Two libraries need to be downloaded and imported into Arduino : AES.h and Decaduino.h. Both are provided into Arduino directory.
For latest version of Decaduino see : https://github.com/irit-irt/DecaDuino

Raspberry:

Raspberry runs Raspbian OS. Client code can be found in client directory. See MQTT section.
Has to be run with python3 (command: python3 client_mqtt.py). Install serial library with pip install (even if it is already there to get the newest version).The teensyduino can be managed directly on the Raspberry Pi as Arduino IDE is available. A zip file for Arduino IDE 1.8.5 is provided in Src directory.

To use Arduino IDE on the Raspberry Pi: Teensyduino packages need to be added into the IDE; zip file can be found on PJRC.
Check /etc/resolv.conf if any problem with WiFi connexion, sometimes wrong server addresses are provided.

Server:
See MQTT section.
MQTT Broker and 3D engine can be run on the same machine if the CPU can handle it.

3D engine:
Download Panda 3D with embedded Python 3.6. Note that regular versions of Panda 3D use Python 2.7, which will not be able to run the codes since it includes Python 3 libraries. Panda 3D with embedded python 3.6 can be found there:
https://www.panda3d.org/download.php?platform=windows&version=devel&sdk
Anchors positions are set in anchors.tab. The multilateration function should be set in the localize method of the World class in World.py.
Important: use the embedded Panda 3D python interpreter rather than interpreters found in regular python installation. On Windows, most convenient way is to add the path to Panda 3D python interpreter before any instance of other interpreters (*typically C:\Panda3D-1.10.0-x64\python*). Add also C:\Panda3D-1.10.0-x64\python\scripts to get pip in your path.

MQTT:
Nodes data are dumped through MQTT protocol (see for more information).
MQTT client: paho (*pip install paho-mqtt* on the Raspberry Pi & server)
MQTT broker: mosquitto (download here https://mosquitto.org/download/). To run (on a Windows laptop): go to Program Files/mosquito directory and run mosquito.exe. *Note: this broker is relatively resources-consuming so you may want to shut it off when not in use to avoid overheating.*

3D engine manual

Before running:
Check doc directory for code documentation.
Install all the libraries needed with pip (*on Windows: py –m pip install library_name*). An error with missing libraries will be raised when trying to run.

To run the 3D engine:
Run python main.py in a shell in the project directory.
Parameters.py contains a wide set of parameters for the type of execution you want.

You will notably need to modify these ones.
ENABLE_LOGS : logs the dataset into json files when enabled.
PLAYBACK: replays the json logs in the playback directory. See Playback section.
MEASURING: starts the engine in measurement mode. See Measurements section. **Note: Measuring should be disabled when using Playback mode.**

Check code documentation for further information on the parameters available.

Playback

This mode allows to replay a scenario that has been recorded previously. The ranging values dumped by the anchors in the json file are replayed, the replay speed can be set from the tkinter menu (See Tkinter section). The positions logged are **NOT** reused and are recomputed, which means that different localization algorithms can be used and compared on the same dataset.
Copy the log file to be replayed in the playback directory and enable PLAYBACK in parameters.py before running.
Note: if several files are in the playback directory, the most recent one will be replayed.

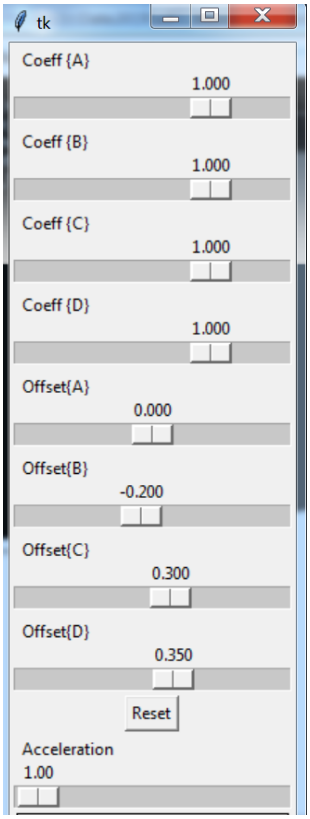
Measurements:

This mode allows characterizing the accuracy of the localization systems on set of reference points with known locations.
First enter the studied reference point in rp.tab. Follow the current formatting of the file. See map section for more information on how to calculate coordinates.
Then, set NB_MEASUREMENTS and NB_REST in parameters.py. The first one defines how much measurements will be done for each reference point. Note that a single ranging made by one anchor is considered as measurement, which means that a full turn of rangings with N anchors equals N measurements. NB_REST defines the number of 'blank' (aka, unlogged) measurements between two reference points, and needs to be set high enough to have the time to move anchor from one point to another.
Every time a measurement for one reference point is a starting/ending a musical signal will notify you, sound should be turned on.
A class for measurements analysis is provided in readMeasurements.py. Check code documentation for further information.

Tkinter

A tkinter Menu allows setting a few parameters in real-time. Corrections can be applied on the ranging from that menu (either as an offset or a coefficient), and in playback mode the playback speed can be increased from that menu as well. Dynamic functionalities should be implemented in the Tkinter. Reset button brings the parameters back to their original values.

Figure 2- Tkinter Menu



Logs

When logs are enabled and/or measurement mode is on, the datasets are logged into json file.
4 sub-directories are used for json logs (in json directory):

- **MQTT:** logs for ranging values in normal mode
- **Pos:** logs for positions in both normal & measurement mode
- **Measurements:** logs for rangings in measurement mode.
- **Playback:** logs for positions in playback mode.

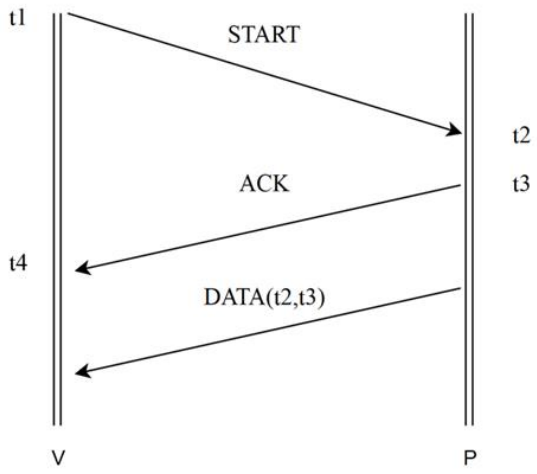
Logs file names are based on the current data & time.

Localization principles

Rangings

SecureLoc system is based on UWB Time-of-Flight localization. The distance between the tag and the anchor is calculated through Two-Way Rangin (TWR) protocol as following:

Figure 3- TWR protocol



The time-of-flight is given by :

$$d(TWR) = \frac{(t4 - t1) - (t3 - t2)}{2} * c$$

The tag is not informed of the distance in the current model.

Ranging filtering

A sliding window is applied on the ranging. This window is a combination of median and mean filtering. Two parameters are given: the **window size (S)** and the **eliminations numbers (E)(E).** Basically, the S distance values in the SW filter are sorted and the E maximum values eliminated. The output of the filter defined as the mean of the S – E values remaining.

Position computation

Several methods are available to compute the tag's position:

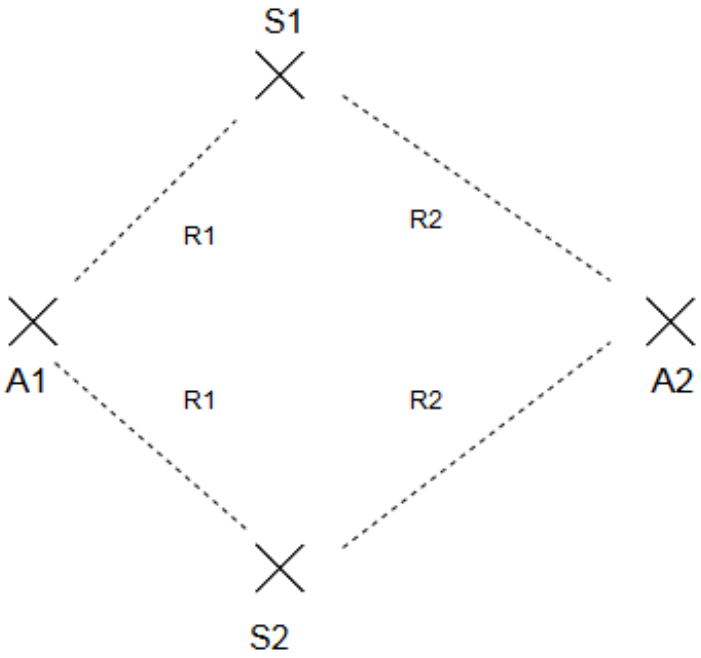
- **Weighted centroid**
- **Gauss-Newton**
- **Weighted-centroid + Iterative MSE reduction**

Weighted centroid

For 2D positioning, two anchors are enough to get a solution.

From a mathematical perspective, applying Pythagora's Threorem with two ranging values lead to two solutions:

Figure 4- Trilateration



If only two anchors are available, the map should be reduced to a single side of the anchors (i.e., either the right of left side to the line (A1,A2)) such as excluding one of the solutions. If other anchors are available, both solutions are compared to the rangings obtained to the other anchors and the closer one is kept.

For each set of 2 anchors among the N anchors available, a position is calculated by trilateration as described above. Then, the Mean-Squared Error of each solution is computed. The MSE of each anchor for a given position P is defined as the square of the difference between the distance measured and the distance between the anchor and P. The MSE of a position P is defined as the sum of the anchors MSE for P. Each trilateration solution receives a coefficient proportional to its MSE and the final solution is defined as the weighted centroid of all trilateration solutions.

Gauss-Newton

Standard Gauss-Newton with a magnitude reduction after each iteration. See for example https://en.wikipedia.org/wiki/Gauss%E2%80%93Newton_algorithm

The starting point can be either the solution of weighted centroid or an arbitrary point (RANDOM_SEARCH).

Weighted-centroid + Iterative MSE reduction

This method first computes the weighted centroid solution, then iterates by small steps from this position to find neighbor point with a lower MSE. The number of iterations and the magnitude of the steps can be set. This method can slightly improve the accuracy of the weighted centroid.