# Technical Documentation

Editor: Pontus Erlesand

**Version 0.1**

Status

| | | |
|---|---|---|
| Reviewed | Pontus Erlesand | 2016-12-14 |
| Approved | | |

# PROJECT IDENTITY

2016/HT, TSKS05-POZYX
Linköping University, (ISY)

Group members

| **Name** | **Responsibility** | **Phone** | **Email** |
|---|---|---|---|
| Rasmus Vilhelmsson(RV) | Project Leader (PL) | 0700 423 400 | rasvi146@student.liu.se |
| Pontus Erlesand(PE) | Head of documentation (DOC) | 0707 964 955 | poner538@student.liu.se |
| Markus Petersson(MP) | Head of sensor fusion | 0703 823 630 | marpe163@student.liu.se |
| Ching-Hsiang Yang(CY) | Head of hardware | 0737 227 489 | chiya469@student.liu.se |
| Susmita Saha(SS) | Head of testing | 0720 451 029 | sussa301@student.liu.se |

**Group e-mail**: pozyx.cdio@gmail.com
**Customer**: ISY, Linköping University, 581 83 Linköping
**Customer contact**: Danyo Danev, 013-281335, danyo.danev@liu.se
**Course leader**: Danyo Danev, 013-281335, danyo.danev@liu.se
**Supervisor**: Trinh Van Chien, 013-282643, trinh.van.chien@liu.se

# Contents

Document history

| Version | Date | Changes | Sign | Reviewed |
|---|---|---|---|---|
| 0.1 | 2016-12-14 | First draft | | PE |
| | | | | |

# 1 Introduction

This document provides a technical overview of the system. In this project, a system was created to perform positioning and tracking of a small electrical device called a tag.

The system uses the positioning and ranging functions provided by the Pozyx platform. MATLAB is used to process data and present the result.

# 2 Definition of terms

| Term | Description |
|---|---|
| 2D | Two dimensions |
| 3D | Three dimensions |
| LOS | Line of sight |
| NLOS | Non line of sight |
| KF | Kalman filter |
| EKF | Extended Kalman filter |
| GUI | Graphical User Interface |
| UWB | Ultra Wide Band |
| USB | Universal Serial Bus |
| TOA | Time of Arrival |

# 3 Overview

## 3.1 System summary

The Pozyx platform provides ranging and positioning using UWB technology, which works both indoors and outdoors. The system requires four anchors to track the tag in 3D, or three anchors if 2D positioning is desired. Within LOS, the UWB signal can cover a maximum range of 100 meters [3], but this is rarely the case for indoor positioning.

To set up the system, small devices called anchors are deployed in the area of interest. The tag communicates with anchors in range using UWB, and estimates the range to each of the anchor. This data is sent to an Arduino board over I$^2$C [4]. The Arduino board in turn transmits the data over a serial connection to a personal computer for processing. The MATLAB application estimates the tag position and displays the estimated position on a GUI. It uses these position estimates to estimate the trajectory of the tag as well.

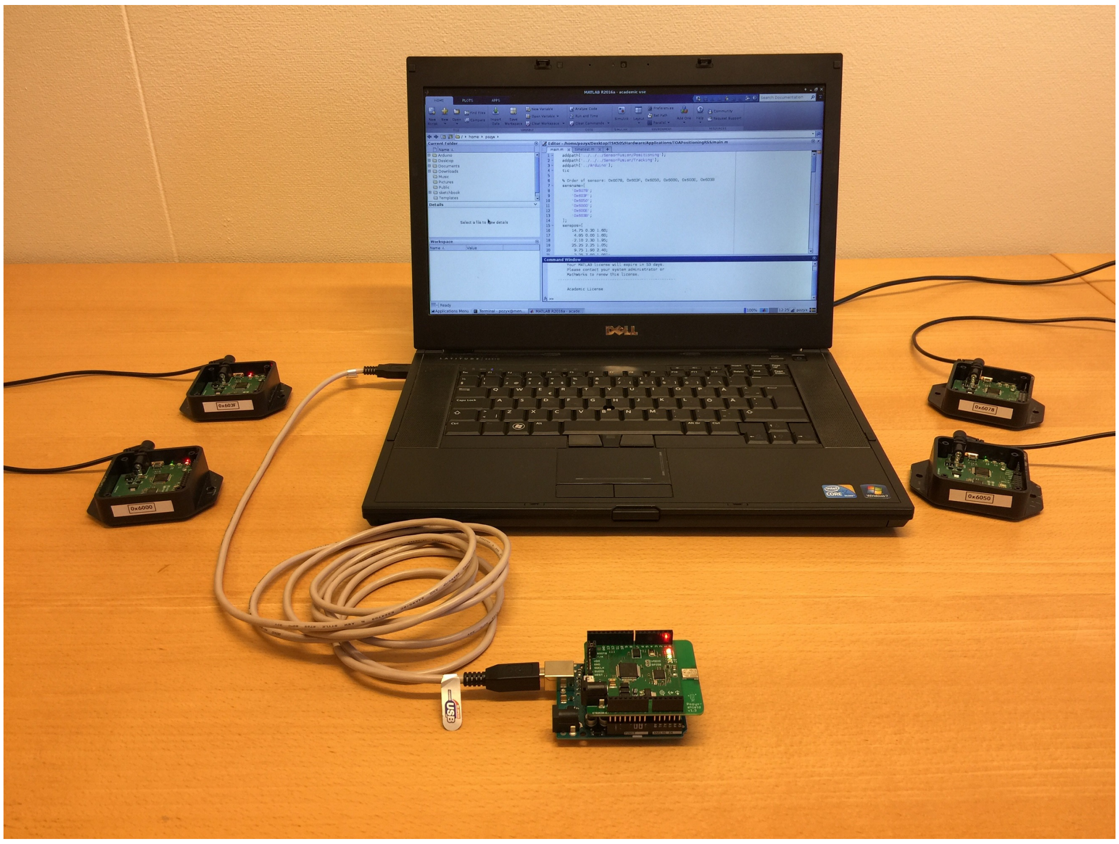


Figure 1: Setup of the Pozyx Platform

## 3.2 Hardware setup

This section provides a general description of the hardware components involved in the system. Figure 1 shows an overview of the Pozyx components.

The utilized hardware components are listed below:

- A personal computer
- An Arduino board
- A Pozyx tag
- Pozyx anchors (Three or more)

### 3.3 Dataflow of the system

The communication between the anchors and the Pozyx tag is through UWB. From the Pozyx shield to the Arduino board the data is transmitted through I$^2$C and finally from the Arduino board to the PC the data is transmitted through serial connection.

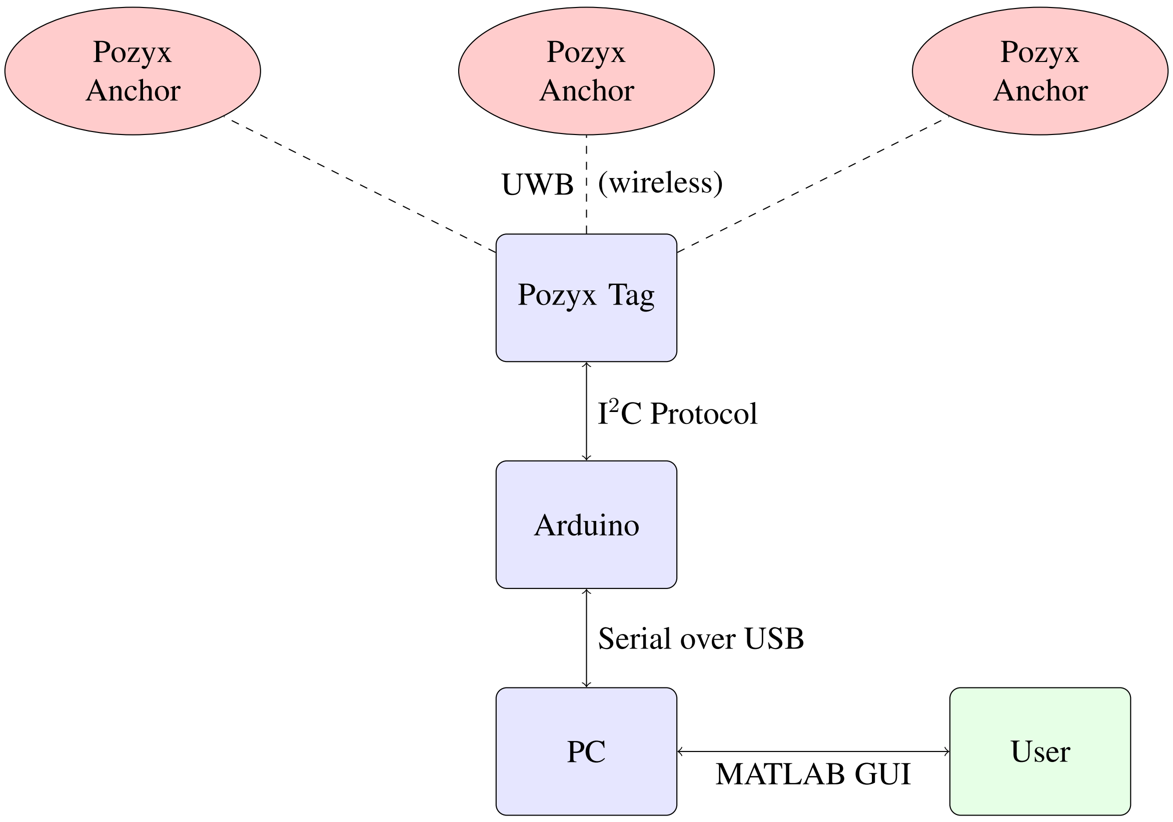


Figure 2: An overview of the system dataflow

## 4 Operating Principle

In this section, the operation principle of the system is discussed.

### 4.1 Hardware

The hardware part of the project consists of the Pozyx platform (including tags and anchors), an Arduino board, and a personal computer with an installation of MATLAB running on Windows.

Pozyx anchors are devices placed in the environment where the tracking/positioning is performed. A minimum of three anchors is required to perform tracking/positioning in 2D and a minimum of four anchors is required in 3D.

A Pozyx tag communicates with the Pozyx anchors via UWB and reports estimates of the range to each anchor. The tag also contains a built-in algorithm that estimates the position of the tag in hardware. The data is sent to the computer through serial connection.

We use a commodity personal computer to perform data processing using MATLAB.

### 4.1.1 Anchor placement

The first step in setting up the system is to deploy anchors in the area of interest. The anchors are powered by USB, but otherwise operate without wired connection. It is required to deploy the anchors at positions so that, for all tag positions, there are at least three anchors with LOS to the tag for proper positioning and tracking. To enable 2D positioning at least three anchors are required in LOS and to enable 3D positioning at least 4 are required.

### 4.1.2 Arduino

An Arduino board is used to provide an interface between the Pozyx tag and the computer. The Pozyx shield is fitted onto the Arduino digital pins, and I$^2$C interface is used to send data from the Pozyx shield to the Arduino board. The Arduino includes a chip to emulate serial communication over USB, so the user can connect the Arduino via USB to the computer for further data processing. In figures 3 and 4 the two components to build a tag is shown.

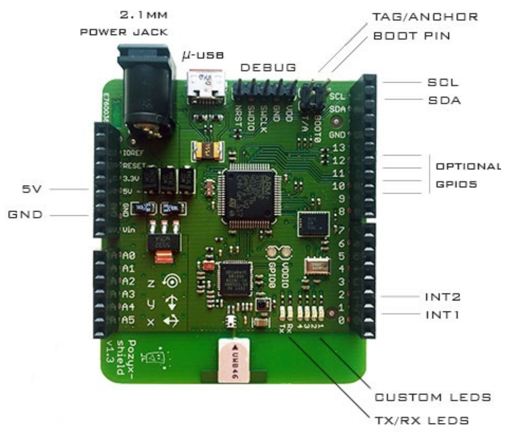


Figure 3: A Pozyx tag [2]

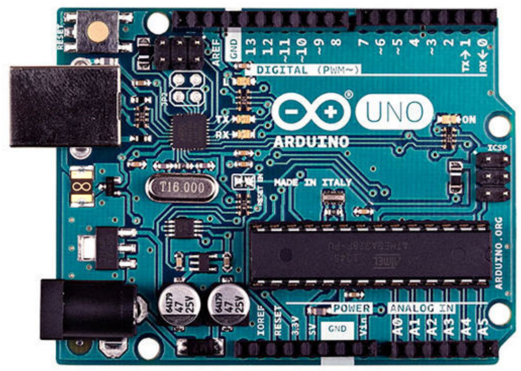


Figure 4: Arduino Board [6]

### 4.1.3 Computer environment

The user should have MATLAB (2016b) and Arduino software installed on the computer. After connecting the Arduino to the computer through USB, the Arduino IDE is used to program the Arduino board.

## 4.2 Software

The system is designed to allow accurate positioning and movement tracking of a Pozyx tag. The software which is written in MATLAB receives data from the tag and performs filtering to obtain better trajectory estimates.

A GUI displays the resulting trajectory in real-time.

## 4.3 Positioning

This section presents how TP$^3$ performs positioning, both using the estimates given by the platform and by solving a non-linear least squares problem.

### 4.3.1 Positioning aided by the Pozyx platform

The Pozyx platform continuously provides estimates of the tag position, no raw data processing required. The calculations are done on the tag.

### 4.3.2 TOA positioning in 3D

This section presents how TOA positioning, based on the ranges provided by the Pozyx platform, is performed.

Let $\boldsymbol{P}$ be the position of the tag, and $\boldsymbol{A_i}$ ($i = 1, ..., n, n \geq 4$) be the positions of the anchors. Also, let $r_i$ be the measured distances from sensor $i$ to the tag, and finally, let $d_i = \text{dist}(\boldsymbol{P}, \boldsymbol{x_i})$. The position of the tag is then estimated as the position $\hat{\boldsymbol{P}}$ that minimizes the sum squared error (SSE), i.e. the solution to the (non-linear) least squares problem:

$$\hat{\boldsymbol{P}} = arg \min_{\boldsymbol{P}} \sum_{i=1}^{n} (d_i - r_i)^2. \tag{1}$$

This optimization problem is solved using the MATLAB optimization toolbox. In order to reduce the chance that the optimization algorithm converges to a local optimum several initial values for the optimization algorithm are used. The calculations are done on the computer.

### 4.3.3 2D Positioning using only three anchors in 3D space

The anchors in the project were deployed in the Communication Systems corridor to enable positioning and tracking in both the coffee room, and large parts of the narrow corridor. Since only six anchors were available for deployment, it was difficult for the tag to obtain good measurements from an adequate number of anchors (four), that is required to do 3D positioning. The way that this was solved was by assuming a fixed height ($z - value$) of the tag. It was now possible to project the measured ranges into this $xy$-plane, and perform 2D positioning, using only 3 anchors.

After projecting the ranges onto the plane, obtaining the ranges $r^i_{\text{xy}} = \sqrt{r_i^2 - z^2_{\text{assumed}}}$, positioning was done in a similar manner as described in equation 1, using the newly obtained ranges $r^i_{\text{xy}}$ and the unknown two dimensional position of the tag $\boldsymbol{P}$.

## 4.4 Tracking

The tracking combines the measurements/estimates of the position with a motion model. This section presents the different filter and motion models TP$^3$ uses to perform tracking.

### 4.4.1 Kalman filter with a constant velocity model

Information about the Kalman Filter can be found e.g. in [1, p. 153]. The (discrete time) constant velocity model is given in [1, p. 344] as

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ \dot{x}_{k+1} \\ \dot{y}_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & T & 0 \\ 0 & 1 & 0 & T \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_k \\ y_k \\ \dot{x}_k \\ \dot{y}_k \end{bmatrix} + \begin{bmatrix} \frac{T^2}{2} & 0 \\ 0 & \frac{T^2}{2} \\ T & 0 \\ 0 & T \end{bmatrix} w_k, \tag{2}$$

where $T$ is the sample time, and $w_k$ is process noise (acceleration in the $x$- and $y$-directions). This is a rather simple model, but has proven itself useful. This is likely due to the fact that it is a quite good model of how people move in corridors.

It is assumed that direct measurements of the position can be obtained, i.e. we have the measurement model

$$m_k = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_k \\ y_k \\ \dot{x}_k \\ \dot{y}_k \end{bmatrix} + e_k, \tag{3}$$

where $e_k$ is measurement noise.

### 4.4.2 Extended Kalman Filter using a coordinated turn model with cartesian coordinates

In section 4.4.1 a linear constant velocity model was considered. However, since it is not reasonable to assume linearity, an EKF is also implemented in order to investigate the difference of these two filtering methods for the given problem.

The EKF utilizes linearization around the current estimated state using a Taylor expansion. In general, the EKF uses both non-linear measurement models and motion models. In the implementation, only a non-linear motion model was considered, since the position estimates are given directly. Thus our system can be described as

$$\begin{aligned} x_{k+1} &= f(x_k, w_k) \\ m_k &= Hx_k + e_k \end{aligned}, \tag{4}$$

where $f$ is a non-linear function of the state $x_k$ and the process noise $w_k$, $H$ is a matrix, and $e_k$ is measurement noise. More information about the EKF can be found in [1, p. 197]

The non-linear motion model, $f$, that we have used in TP$^3$ is a coordinated turn model in cartesian coordinates. This model is described in [1, p. 353]. The measurement model used is identical to the one given in equation 3.

### 4.4.3 Trajectory estimation

The estimated trajectory is obtained by taking the position coordinates from the calculated state after the measurement update in the KF/EKF filter, in each iteration.

Since the GUI only shows $x$ and $y$ positions the trajectory only needs to save the first two elements in each state, i.e.

$$TRJ(k) = \begin{bmatrix} x_k \\ y_k \end{bmatrix}$$

where $TRJ(k)$, $k = 1, 2, \ldots$ is the current position, $(x_k, y_k)$, at time $k$.

### 4.4.4 Smoothing the trajectory

When tracking is done using the described methods the resulting trajectory is often jagged, since the measurements contain noise. In order to get an estimated trajectory that is less rough, smoothing can be done after e.g. the Kalman filter estimates the trajectory. The system has several filtering options for smoothing (all different low pass filters). The different Low pass filters are:

- Low-Pass Butterworth filtering (Optimally flat pass band)
- Low-Pass Chebyshev (1 and 2) (Optimally narrow transition band)
- Moving average

More information about Butterworth and Chebyshev filters can be found in [5, chap.4.10]. The smoothing using moving average is performed as in equation 5.

$$\boldsymbol{x}_i^{ma} = \frac{x_i + x_{i-1} + \cdots + x_{i-n}}{n} \tag{5}$$

where $\boldsymbol{x}^{ma}$ is the calculated moving average for the current point $(x, y)$ and $\boldsymbol{x}_i$, $i = 1, 2, .., n$ are the obtained position estimates. This means that $n$ symbolizes the order of the moving average smoothing. Appropriate orders for the moving average are between $n = 10$ and $n = 15$.

The order of the Butterworth and Chebyshev are set by default to an order of 5 in the system. The cutoff frequency is chosen by the user, appropriate values for the (normalized) cutoff frequency is between $0.05$ and $0.3$.

## 4.5 Classes for data processing

In the table below, all classes used in the data processing module are presented, along with a short description of each class.

Table of classes for data processing

| Class | Description |
|---|---|
| `kalmantracker` | Kalman filter, which is a property of the tracker class. |
| `ekftracker` | Extended Kalman filter, an alternative filter in the tracker class. |
| `tracker` | Handles the tracking of the tag. |
| `trajectory` | Handles the trajectory, which is a property of the tracker class. In this class the smoothing of the trajectory is also handled. |

## 4.6 GUI

In this section, a description for the GUI and its functionality is presented. The GUI is made in MATLAB using GUIDE, a built-in interactive development environment. Object-oriented programming in MATLAB is used to develop the GUI the data processing module.

The GUI module includes three classes, these are for drawing a map, displaying the anchor positions and displaying the tag position. A description of these classes can be found in the table below.

Table of classes for the GUI

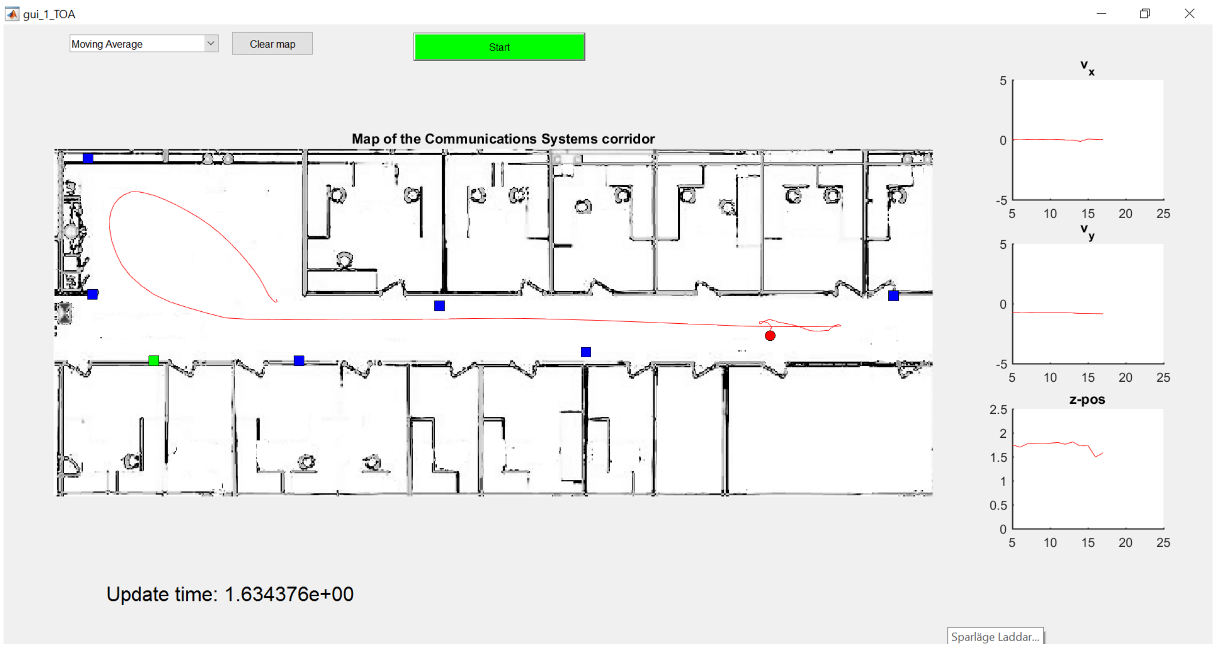
| Class | Description |
|---|---|
| `circle` | Draws a circle on indicate where the tag is |
| `anchor` | Draws the position of the anchors on the map |
| `map` | Draws the map, anchors and estimated position of the tag |

Before the GUI is initialized, the program presents an input dialog box where the user has to fill in information about the anchors and the tag. This includes the number of anchors used, the type of filter used for smoothing, cutoff frequency for smoothing and moving average order. If the user wants to use the default values he/she only has to press the OK-button in the dialog box.

When the GUI is initialized, objects for fetching data from the Arduino, processing data and drawing on a map, are created. Pointers to these objects are created as well. The GUI then waits for the user to mark the positions of the anchors on the map, unless default values are chosen. The first anchor placed will be the origin. It is important to fix the origin at the right place on the map so it will match the origin programmed in the Arduino board, otherwise the position displayed on the map will be incorrect.

After the anchor placement, the program waits for the user to click on the toggle-button labeled `Start`. When this button is pressed the program enters the main loop and starts performing positioning and tracking. There is a drop down menu on the GUI that lets the user switch filter type for smoothing the trajectory. The user can choose between a Butterworth filter, two types of Chebyshev filters or estimation using moving average. The filter can be changed at any time after the program has been initialized but not when the system is running and estimating the positions and trajectories.

Next to the map on the GUI, there are three plots that present some additional information about the tag, the velocities in $x$ and $y$ directions, and the altitude of the tag. Below the map in the GUI, there is a text box displaying how often the position is updated on the map. Figure 5 illustrates the GUI when it's running.



Update time: 1.634376e+00

Figure 5: The systems GUI when the program is running

## References

[1] Fredrik Gustafsson. *Statistical Sensor Fusion*. Lund: Studentlitteratur AB, 2010. ISBN: 9789144077321.

[2] Pozyx Labs. *Datasheet – Pin-out*. `https://www.pozyx.io/Documentation/Datasheet/SystemDescription/pinOut`. Retrieved 13-December-2016.

[3] Pozyx Labs. *Frequently Asked Questions*. `https://www.pozyx.io/Documentation`. Retrieved 13-December-2016.

[4] Pozyx Labs. *System Description*. `https://www.pozyx.io/Documentation/Datasheet/SystemDescription`. Retrieved 13-December-2016.

[5] B. P. Lathi. *Linear Systems and Signals Second Edition*. 2010. Chap. 4. ISBN: 9780195392562.

[6] Arduino Srl. *Arduino UNO*. `http://www.arduino.org/products/boards/arduino-uno`. Retrieved 13-December-2016.