Министерство образования и науки РФ

Федеральное государственное бюджетное образовательное учреждение

высшего образования «Московский политехнический университет»

**факультет информационных технологий**

**Кафедра СМАРТ-технологий**

Дисциплина: Разработка систем сбора и обработки данных

Отчёт по лабораторной работе №1

«Обработка линейно структурированных данных»

Работа выполнена (ФИО):

Ф И О

Научный руководитель

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ф И О

**Цель работы:**

Разработать алгоритм считывания и визуализации потоковых данных с

плоскостного 2D-лидара

**Задачи:**

* Подготовить приложение считывания данных плоскостного лидара, сохраненных в потоковом формате;
* Разработать функцию покадровой визуализации данных фреймов лидара;
* Разработать функцию детектирования объектов методом кластерного анализа;
* Разработать метод отслеживания траекторий объектов между фреймами;

**Порядок выполнения работы**

Задача 1

Подготовить приложение для считывания данных из файла, хранящего поток данных с

плоскостного лидара. Представить возможность просмотра данных в текстовом виде как в

непрерывном виде (отображается покадрово в соответствии с временными отметками) с

возможностью поставить чтение «на паузу» (будет отображаться один фрейм).

Задача 2

Подготовить систему визуализации данных лидара в виде круговой диаграммы

(данные приходят в мм). Использовать небольшие квадраты или окружности для

отображения позиций препятствий (концов лучей).

Задача 3

Разработать функцию поиска центров масс кластеров методом k-means и отсечения

линейных объектов (стен) за счет разбиения на блоки с настраиваемым количеством

точек. Выполнить обводку прямоугольниками найденных объектов.

Детектирование линейных объектов выполняется путем поиска цепочек объектов,

выстроенных в одну линию.

Задача 4

Для отдельных объектов и краевых объектов (концов стен) выполнить поиск

перемещения объектов между кадрами. Отрисовывать траектории движения центров

объектов поверх изображения текущего кадра. Координаты центров хранить в списках и

выводить в структурированном виде в listbox. **Ход работы**

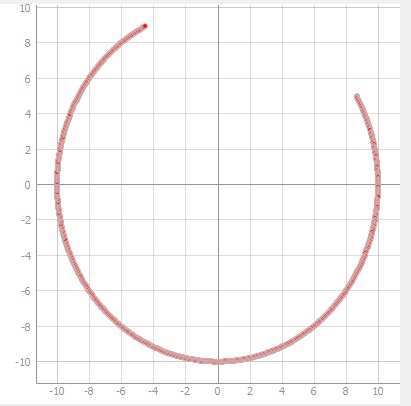
В качестве источника данных выступает дамп с 2D лидара имеющий следующую структуру:

12:09:12.80> 19 19 … 647 640 19

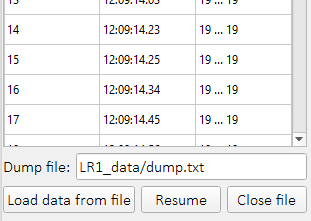
…

12:09:24.20> 19 19 … 790 764 76

Дамп содержит не все 360 градусов обзора. Из него как бы «вырезана» часть обзора. Реальная картина выглядит примерно так:

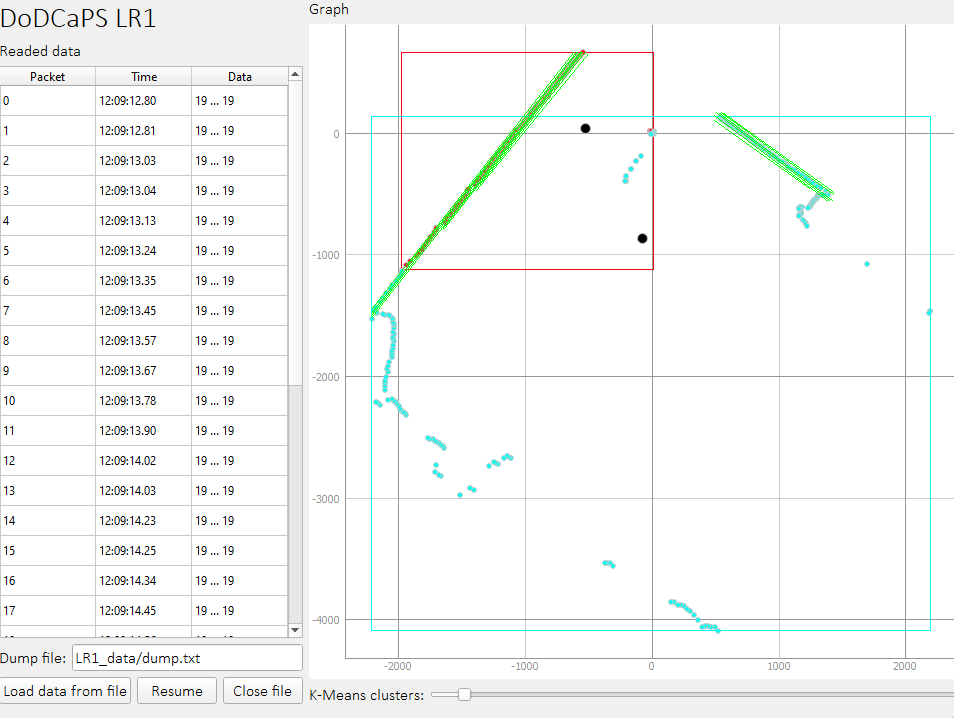


Для начала чтения файла, на форме необходимо указать расположение дампа и нажать кнопку Load data from file. В процессе, чтение можно поставить на паузу и продолжить с помощью кнопки Pause / Resume. Также, чтение можно оставить и закрыть файл, нажав на кнопку Close file.



В процессе чтения данные в реальном времени отображаются в таблице. Первая колонка – номер пакета, вторая – время, третья – первое и последнее число из дампа.

После чтения данные проходят через K-means и обрисовываются в виде облаков точек. А также, с помощью преобразования Хафа вычисляются точки, лежащие на одной линии, и выделяются зелёным цветом.



Файл парсится построчно. Далее каждая строка преобразуется в массив, разделитель – символ пробела. Из самого первого элемента убирается символ ‘>’, в итоге, остаётся штамп времени. Разница между штампами времени передаётся в функцию time.sleep() для того, чтобы имитировать чтение дампа в реальном времени.

Координаты X и Y из массива точек определяются следующим методом

angle = (60.0 + i \* 0.36) \* math.pi / 180  
  
x = data\_packet[i] \* math.sin(angle)  
y = data\_packet[i] \* math.cos(angle)

i – счётчик, изменяется от 0 до количества чисел в пакете.

Для поиска линий, по точкам строится изображение в формате массива NumPy, которое в дальнейшем передаётся в функцию cv2.HoughLinesP().

**Вывод**

В ходе данной работы было создано приложение, реализующее алгоритм считывания и визуализации потоковых данных с плоскостного 2D-лидара

**Исходный код**

*"""  
This is free and unencumbered software released into the public domain.  
  
Anyone is free to copy, modify, publish, use, compile, sell, or  
distribute this software, either in source code form or as a compiled  
binary, for any purpose, commercial or non-commercial, and by any  
means.  
  
In jurisdictions that recognize copyright laws, the author or authors  
of this software dedicate any and all copyright interest in the  
software to the public domain. We make this dedication for the benefit  
of the public at large and to the detriment of our heirs and  
successors. We intend this dedication to be an overt act of  
relinquishment in perpetuity of all present and future rights to this  
software under copyright law.  
  
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,  
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF  
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.  
IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY CLAIM, DAMAGES OR  
OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,  
ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR  
OTHER DEALINGS IN THE SOFTWARE.  
  
For more information, please refer to <https://unlicense.org>  
"""*import datetime  
import math  
import os  
import sys  
import threading  
import time  
  
import cv2  
import matplotlib.pyplot as plt  
import numpy as np  
import pyqtgraph as pg  
from PyQt5 import uic, QtWidgets, QtCore  
from PyQt5.QtWidgets import QApplication, QMainWindow, QTableWidgetItem  
  
  
class KMeans:  
 *"""  
 K-means clustering  
 code from: https://dev.to/rishitdagli/build-k-means-from-scratch-in-python-2140  
 """* def \_\_init\_\_(self, k=2, tol=0.001, max\_iter=300):  
 self.k = k  
 self.tol = tol  
 self.max\_iter = max\_iter  
  
 def fit(self, data, k=2, tol=0.001, max\_iter=300):  
 self.k = k  
 self.tol = tol  
 self.max\_iter = max\_iter  
 self.centroids = {}  
  
 for i in range(self.k):  
 self.centroids[i] = data[i]  
  
 for i in range(self.max\_iter):  
 self.classifications = {}  
  
 for i in range(self.k):  
 self.classifications[i] = []  
  
 for featureset in data:  
 distances = [np.linalg.norm(featureset - self.centroids[centroid]) for centroid in self.centroids]  
 classification = distances.index(min(distances))  
 self.classifications[classification].append(featureset)  
  
 prev\_centroids = dict(self.centroids)  
  
 for classification in self.classifications:  
 self.centroids[classification] = np.average(self.classifications[classification], axis=0)  
  
 optimized = True  
  
 for c in self.centroids:  
 original\_centroid = prev\_centroids[c]  
 current\_centroid = self.centroids[c]  
 if np.sum((current\_centroid - original\_centroid) / original\_centroid \* 100.0) > self.tol:  
 print(np.sum((current\_centroid - original\_centroid) / original\_centroid \* 100.0))  
 optimized = False  
  
 if optimized:  
 break  
  
 def predict(self, data):  
 distances = [np.linalg.norm(data - self.centroids[centroid]) for centroid in self.centroids]  
 classification = distances.index(min(distances))  
 return classification  
  
  
class Window(QMainWindow):  
 def \_\_init\_\_(self):  
 super(Window, self).\_\_init\_\_()  
 *# Load GUI file* uic.loadUi(**'LR1.ui'**, self)  
  
 *# System variables* self.model = KMeans()  
 self.dump\_file = None  
 self.reader\_running = False  
 self.dump\_paused = False  
 self.points = []  
  
 *# Connect GUI controls* self.btn\_load\_data.clicked.connect(self.load\_data)  
 self.btn\_stop\_reading.clicked.connect(self.stop\_reading)  
 self.btn\_pause.clicked.connect(self.pause)  
 self.plot\_timer = QtCore.QTimer()  
 self.plot\_timer.timeout.connect(self.update\_plot)  
 self.plot\_timer.start(100)  
  
 *# Initialize table* self.init\_tables()  
  
 *# Initialize pyQtGraph charts* self.init\_charts()  
  
 *# Show GUI* self.show()  
  
 def init\_tables(self):  
 *"""  
 Initializes table of packets and setup table (whitelist table)  
 :return:  
 """* self.points\_table.setColumnCount(3)  
 self.points\_table.verticalHeader().setVisible(False)  
 self.points\_table.setEditTriggers(QtWidgets.QAbstractItemView.NoEditTriggers)  
 self.points\_table.setHorizontalHeaderItem(0, QtWidgets.QTableWidgetItem(**'Packet'**))  
 self.points\_table.setHorizontalHeaderItem(1, QtWidgets.QTableWidgetItem(**'Time'**))  
 self.points\_table.setHorizontalHeaderItem(2, QtWidgets.QTableWidgetItem(**'Data'**))  
 header = self.points\_table.horizontalHeader()  
 header.setSectionResizeMode(0, QtWidgets.QHeaderView.Stretch)  
 header.setSectionResizeMode(1, QtWidgets.QHeaderView.Stretch)  
 header.setSectionResizeMode(2, QtWidgets.QHeaderView.Stretch)  
  
 def init\_charts(self):  
 *"""  
 Initializes charts  
 :return:  
 """* self.graphWidget.setBackground((255, 255, 255))  
 self.graphWidget.showGrid(x=True, y=True, alpha=1.0)  
  
 def update\_plot(self):  
 *"""  
 Draws points over pyQTGraph  
 :return:  
 """* if len(self.points) > 0 and not self.dump\_paused:  
 self.graphWidget.clear()  
  
 *# Find K-means clusters* self.model.fit(np.array(self.points), k=self.slider\_clusters.value())  
  
 *# Draw centroids* centroids\_x = []  
 centroids\_y = []  
 for centroid in self.model.centroids:  
 centroids\_x.append(self.model.centroids[centroid][0])  
 centroids\_y.append(self.model.centroids[centroid][1])  
  
 *# Draw points by clusters* color\_data = np.array(range(len(self.model.classifications) + 1))  
 color\_map = plt.get\_cmap(**'hsv'**)  
 min\_z = np.min(color\_data)  
 max\_z = np.max(color\_data)  
 rgba\_img = color\_map(1.0 - (color\_data - min\_z) / (max\_z - min\_z)) \* 255  
 for classification in self.model.classifications:  
 features\_x = []  
 features\_y = []  
 for features\_et in self.model.classifications[classification]:  
 features\_x.append(features\_et[0])  
 features\_y.append(features\_et[1])  
 self.graphWidget.plot(features\_x, features\_y, pen=None,  
 symbolBrush=(rgba\_img[classification][0],  
 rgba\_img[classification][1],  
 rgba\_img[classification][2]), symbolSize=5)  
  
 max\_x = np.max(features\_x)  
 min\_x = np.min(features\_x)  
 max\_y = np.max(features\_y)  
 min\_y = np.min(features\_y)  
  
 self.graphWidget.plot([min\_x, min\_x, max\_x, max\_x, min\_x], [min\_y, max\_y, max\_y, min\_y, min\_y],  
 pen=pg.mkPen(((  
 rgba\_img[classification][0],  
 rgba\_img[classification][1],  
 rgba\_img[classification][2]))),  
 symbolBrush=None, symbolSize=0)  
  
 *# Plot centroids* self.graphWidget.plot(centroids\_x, centroids\_y, pen=None,  
 symbolBrush=(0, 0, 0), symbolSize=10)  
  
 *# Found lines and draw it* points\_x = np.array([item[0] for item in self.points])  
 points\_y = np.array([item[1] for item in self.points])  
 min\_x = np.min(points\_x)  
 min\_y = np.min(points\_y)  
 points\_x -= min\_x  
 points\_y -= min\_y  
 points\_x = points\_x / 10  
 points\_y = points\_y / 10  
 points\_image = np.zeros((int(np.max(points\_y) + 1), int(np.max(points\_x) + 1)), np.uint8)  
 for i in range(len(points\_x)):  
 points\_image[int(points\_y[i]), int(points\_x[i])] = 255  
 kernel = np.ones((5, 5), np.uint8)  
 points\_image = cv2.dilate(points\_image, kernel, iterations=2)  
 min\_line\_length = 550  
 max\_line\_gap = 70  
 lines = cv2.HoughLinesP(points\_image, 1, np.pi / 180, 100, min\_line\_length, max\_line\_gap)  
 for line in lines:  
 for x1, y1, x2, y2 in line:  
 self.graphWidget.plot([x1 \* 10 + min\_x, x2 \* 10 + min\_x], [y1 \* 10 + min\_y, y2 \* 10 + min\_y],  
 pen=pg.mkPen((0, 255, 0)),  
 symbolBrush=None, symbolSize=0)  
  
 def load\_data(self):  
 *"""  
 Loads dump file  
 :return:  
 """* if not self.reader\_running:  
 if os.path.exists(self.data\_file.text()):  
 print(**'Loading data...'**)  
 self.dump\_file = open(self.data\_file.text(), **'r'**)  
 self.reader\_running = True  
 thread = threading.Thread(target=self.dump\_reader)  
 thread.start()  
 else:  
 print(**'File'**, self.data\_file.text(), **'doesn**\'**t exist!'**)  
  
 def pause(self):  
 *"""  
 Pauses data stream  
 :return:  
 """* self.dump\_paused = not self.dump\_paused  
 if self.dump\_paused:  
 self.btn\_pause.setText(**'Resume'**)  
 else:  
 self.btn\_pause.setText(**'Pause'**)  
  
 def stop\_reading(self):  
 *"""  
 Stops reading data from dump file  
 :return:  
 """* self.reader\_running = False  
 self.dump\_file.close()  
  
 def dump\_reader(self):  
 *"""  
 Reads dump from file  
 :return:  
 """  
 # Clear table and data arrays* self.points\_table.setRowCount(0)  
  
 *# Create variables* packets\_read = 0  
 last\_packet\_datetime = None  
  
 *# Continue reading* while self.reader\_running:  
 *# If on pause* while self.dump\_paused:  
 time.sleep(0.1)  
  
 *# Read line from file* line = self.dump\_file.readline()  
  
 *# Check for line* if line is None or len(line) < 1:  
 break  
  
 data\_packet = line.split(**' '**)  
  
 *# Sleep defined time* time\_string = str(data\_packet[0]).replace(**'>'**, **''**)  
 if last\_packet\_datetime is None:  
 last\_packet\_datetime = datetime.datetime.strptime(time\_string, **'%H:%M:%S.%f'**)  
 packet\_datetime = datetime.datetime.strptime(time\_string, **'%H:%M:%S.%f'**)  
 time.sleep((packet\_datetime - last\_packet\_datetime).total\_seconds())  
 last\_packet\_datetime = packet\_datetime  
  
 *# Add packet to the table* position = self.points\_table.rowCount()  
 self.points\_table.insertRow(position)  
 self.points\_table.setItem(position, 0, QTableWidgetItem(str(position)))  
 self.points\_table.setItem(position, 1, QTableWidgetItem(str(time\_string)))  
  
 *# Remove timestamp and ending from packet and convert to int* data\_packet = list(map(int, data\_packet[1:][:-1]))  
  
 self.points\_table.setItem(position, 2, QTableWidgetItem(str(data\_packet[0]) +  
 **' ... '** + str(data\_packet[-1])))  
  
 points = []  
  
 for i in range(len(data\_packet)):  
 angle = (60.0 + i \* 0.36) \* math.pi / 180  
  
 x = data\_packet[i] \* math.sin(angle)  
 y = data\_packet[i] \* math.cos(angle)  
 points.append([x, y])  
  
 self.points = points.copy()  
  
 *# Increment counter* packets\_read += 1  
  
 self.dump\_file.close()  
 print(**'File reading stopped. Read'**, packets\_read, **'packets'**)  
  
  
if \_\_name\_\_ == **'\_\_main\_\_'**:  
 app = QApplication(sys.argv)  
 app.setStyle(**'fusion'**)  
 win = Window()  
 sys.exit(app.exec\_())

*<?*xml version="1.0" encoding="UTF-8"*?>  
<!--  
This is free and unencumbered software released into the public domain.  
  
Anyone is free to copy, modify, publish, use, compile, sell, or  
distribute this software, either in source code form or as a compiled  
binary, for any purpose, commercial or non-commercial, and by any  
means.  
  
In jurisdictions that recognize copyright laws, the author or authors  
of this software dedicate any and all copyright interest in the  
software to the public domain. We make this dedication for the benefit  
of the public at large and to the detriment of our heirs and  
successors. We intend this dedication to be an overt act of  
relinquishment in perpetuity of all present and future rights to this  
software under copyright law.  
  
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,  
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF  
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.  
IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY CLAIM, DAMAGES OR  
OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,  
ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR  
OTHER DEALINGS IN THE SOFTWARE.  
  
For more information, please refer to <https://unlicense.org>  
 -->*<ui version="4.0">  
 <class>MainWindow</class>  
 <widget class="QMainWindow" name="MainWindow">  
 <property name="geometry">  
 <rect>  
 <x>0</x>  
 <y>0</y>  
 <width>724</width>  
 <height>516</height>  
 </rect>  
 </property>  
 <property name="font">  
 <font>  
 <family>Calibri Light</family>  
 <pointsize>12</pointsize>  
 </font>  
 </property>  
 <property name="windowTitle">  
 <string>DoDCaPS LR3</string>  
 </property>  
 <widget class="QWidget" name="centralwidget">  
 <layout class="QGridLayout" name="gridLayout">  
 <item row="0" column="0">  
 <widget class="QLabel" name="label">  
 <property name="sizePolicy">  
 <sizepolicy hsizetype="Fixed" vsizetype="Fixed">  
 <horstretch>0</horstretch>  
 <verstretch>0</verstretch>  
 </sizepolicy>  
 </property>  
 <property name="font">  
 <font>  
 <pointsize>22</pointsize>  
 </font>  
 </property>  
 <property name="text">  
 <string>DoDCaPS LR1</string>  
 </property>  
 </widget>  
 </item>  
 <item row="0" column="1" rowspan="3">  
 <layout class="QVBoxLayout" name="verticalLayout\_4">  
 <item>  
 <widget class="QLabel" name="label\_26">  
 <property name="sizePolicy">  
 <sizepolicy hsizetype="Preferred" vsizetype="Fixed">  
 <horstretch>0</horstretch>  
 <verstretch>0</verstretch>  
 </sizepolicy>  
 </property>  
 <property name="text">  
 <string>Graph</string>  
 </property>  
 </widget>  
 </item>  
 <item>  
 <widget class="PlotWidget" name="graphWidget" native="true">  
 <property name="sizePolicy">  
 <sizepolicy hsizetype="Expanding" vsizetype="Expanding">  
 <horstretch>0</horstretch>  
 <verstretch>0</verstretch>  
 </sizepolicy>  
 </property>  
 <property name="minimumSize">  
 <size>  
 <width>400</width>  
 <height>400</height>  
 </size>  
 </property>  
 <property name="maximumSize">  
 <size>  
 <width>16777215</width>  
 <height>16777215</height>  
 </size>  
 </property>  
 </widget>  
 </item>  
 <item>  
 <layout class="QHBoxLayout" name="horizontalLayout\_3">  
 <item>  
 <widget class="QLabel" name="label\_3">  
 <property name="sizePolicy">  
 <sizepolicy hsizetype="Fixed" vsizetype="Fixed">  
 <horstretch>0</horstretch>  
 <verstretch>0</verstretch>  
 </sizepolicy>  
 </property>  
 <property name="text">  
 <string>K-Means clusters:</string>  
 </property>  
 </widget>  
 </item>  
 <item>  
 <widget class="QSlider" name="slider\_clusters">  
 <property name="minimum">  
 <number>1</number>  
 </property>  
 <property name="maximum">  
 <number>20</number>  
 </property>  
 <property name="value">  
 <number>2</number>  
 </property>  
 <property name="orientation">  
 <enum>Qt::Horizontal</enum>  
 </property>  
 </widget>  
 </item>  
 </layout>  
 </item>  
 </layout>  
 </item>  
 <item row="1" column="0">  
 <layout class="QVBoxLayout" name="verticalLayout\_2">  
 <item>  
 <widget class="QLabel" name="label\_4">  
 <property name="sizePolicy">  
 <sizepolicy hsizetype="Fixed" vsizetype="Fixed">  
 <horstretch>0</horstretch>  
 <verstretch>0</verstretch>  
 </sizepolicy>  
 </property>  
 <property name="text">  
 <string>Readed data</string>  
 </property>  
 </widget>  
 </item>  
 <item>  
 <widget class="QTableWidget" name="points\_table">  
 <property name="sizePolicy">  
 <sizepolicy hsizetype="Minimum" vsizetype="Expanding">  
 <horstretch>0</horstretch>  
 <verstretch>0</verstretch>  
 </sizepolicy>  
 </property>  
 <property name="sortingEnabled">  
 <bool>true</bool>  
 </property>  
 <attribute name="horizontalHeaderShowSortIndicator" stdset="0">  
 <bool>true</bool>  
 </attribute>  
 <attribute name="verticalHeaderShowSortIndicator" stdset="0">  
 <bool>false</bool>  
 </attribute>  
 </widget>  
 </item>  
 </layout>  
 </item>  
 <item row="2" column="0">  
 <layout class="QVBoxLayout" name="verticalLayout">  
 <item>  
 <layout class="QHBoxLayout" name="horizontalLayout">  
 <item>  
 <widget class="QLabel" name="label\_19">  
 <property name="sizePolicy">  
 <sizepolicy hsizetype="Fixed" vsizetype="Fixed">  
 <horstretch>0</horstretch>  
 <verstretch>0</verstretch>  
 </sizepolicy>  
 </property>  
 <property name="text">  
 <string>Dump file:</string>  
 </property>  
 <property name="alignment">  
 <set>Qt::AlignCenter</set>  
 </property>  
 </widget>  
 </item>  
 <item>  
 <widget class="QLineEdit" name="data\_file">  
 <property name="sizePolicy">  
 <sizepolicy hsizetype="Preferred" vsizetype="Fixed">  
 <horstretch>0</horstretch>  
 <verstretch>0</verstretch>  
 </sizepolicy>  
 </property>  
 <property name="text">  
 <string>LR1\_data/dump.txt</string>  
 </property>  
 </widget>  
 </item>  
 </layout>  
 </item>  
 <item>  
 <layout class="QHBoxLayout" name="horizontalLayout\_2">  
 <item>  
 <widget class="QPushButton" name="btn\_load\_data">  
 <property name="text">  
 <string>Load data from file</string>  
 </property>  
 </widget>  
 </item>  
 <item>  
 <widget class="QPushButton" name="btn\_pause">  
 <property name="text">  
 <string>Pause</string>  
 </property>  
 </widget>  
 </item>  
 <item>  
 <widget class="QPushButton" name="btn\_stop\_reading">  
 <property name="sizePolicy">  
 <sizepolicy hsizetype="Preferred" vsizetype="Fixed">  
 <horstretch>0</horstretch>  
 <verstretch>0</verstretch>  
 </sizepolicy>  
 </property>  
 <property name="text">  
 <string>Close file</string>  
 </property>  
 </widget>  
 </item>  
 </layout>  
 </item>  
 </layout>  
 </item>  
 </layout>  
 </widget>  
 <widget class="QMenuBar" name="menubar">  
 <property name="geometry">  
 <rect>  
 <x>0</x>  
 <y>0</y>  
 <width>724</width>  
 <height>21</height>  
 </rect>  
 </property>  
 </widget>  
 <widget class="QStatusBar" name="statusbar"/>  
 </widget>  
 <customwidgets>  
 <customwidget>  
 <class>PlotWidget</class>  
 <extends>QWidget</extends>  
 <header>pyqtgraph</header>  
 <container>1</container>  
 </customwidget>  
 </customwidgets>  
 <resources/>  
 <connections/>  
</ui>

/\*

This is free and unencumbered software released into the public domain.

Anyone is free to copy, modify, publish, use, compile, sell, or

distribute this software, either in source code form or as a compiled

binary, for any purpose, commercial or non-commercial, and by any

means.

In jurisdictions that recognize copyright laws, the author or authors

of this software dedicate any and all copyright interest in the

software to the public domain. We make this dedication for the benefit

of the public at large and to the detriment of our heirs and

successors. We intend this dedication to be an overt act of

relinquishment in perpetuity of all present and future rights to this

software under copyright law.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,

EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF

MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.

IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY CLAIM, DAMAGES OR

OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,

ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR

OTHER DEALINGS IN THE SOFTWARE.

For more information, please refer to <https://unlicense.org>

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Serial port setup \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Serial port speed

const uint16\_t PORT\_BAUD\_RATE PROGMEM = 57600;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Pins setup \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// CH1 analog pin

const uint8\_t CHANNEL\_1\_PIN PROGMEM = A0;

// CH2 analog pin

const uint8\_t CHANNEL\_2\_PIN PROGMEM = A1;

// CH3 analog pin

const uint8\_t CHANNEL\_3\_PIN PROGMEM = A2;

// CH4 analog pin

const uint8\_t CHANNEL\_4\_PIN PROGMEM = A3;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Time settings \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// After this time in milliseconds the buffer will be pushed to the serial port

// Set to 0 to send packets immediately

const uint16\_t SERIAL\_PERIOD PROGMEM = 10;

// System variables

uint8\_t buffer[11];

uint8\_t i;

uint16\_t channel\_1, channel\_2, channel\_3, channel\_4;

uint64\_t serial\_timer;

void setup()

{

// Init serial port

Serial.begin(PORT\_BAUD\_RATE);

// Pre-define buffer ending

buffer[9] = 255;

buffer[10] = 255;

}

void loop()

{

// Read ADC values

read\_channels();

// Fill buffer

fill\_buffer();

if (millis() - serial\_timer > SERIAL\_PERIOD) {

// Push buffer to the serial port

Serial.write(buffer, 11);

// Reset timer

serial\_timer = millis();

}

}

/// <summary>

/// Fills buffer with 4 ADC values

/// </summary>

void fill\_buffer(void) {

// Fill with channel values

buffer[0] = channel\_1 >> 8;

buffer[1] = channel\_1;

buffer[2] = channel\_2 >> 8;

buffer[3] = channel\_2;

buffer[4] = channel\_3 >> 8;

buffer[5] = channel\_3;

buffer[6] = channel\_4 >> 8;

buffer[7] = channel\_4;

// Calculate checksum

buffer[8] = 0;

for (i = 0; i <= 7; i++)

buffer[8] ^= buffer[i];

}

/// <summary>

/// Reads values from analog inputs

/// </summary>

void read\_channels(void) {

channel\_1 = analogRead(CHANNEL\_1\_PIN);

channel\_2 = analogRead(CHANNEL\_2\_PIN);

channel\_3 = analogRead(CHANNEL\_3\_PIN);

channel\_4 = analogRead(CHANNEL\_4\_PIN);

}