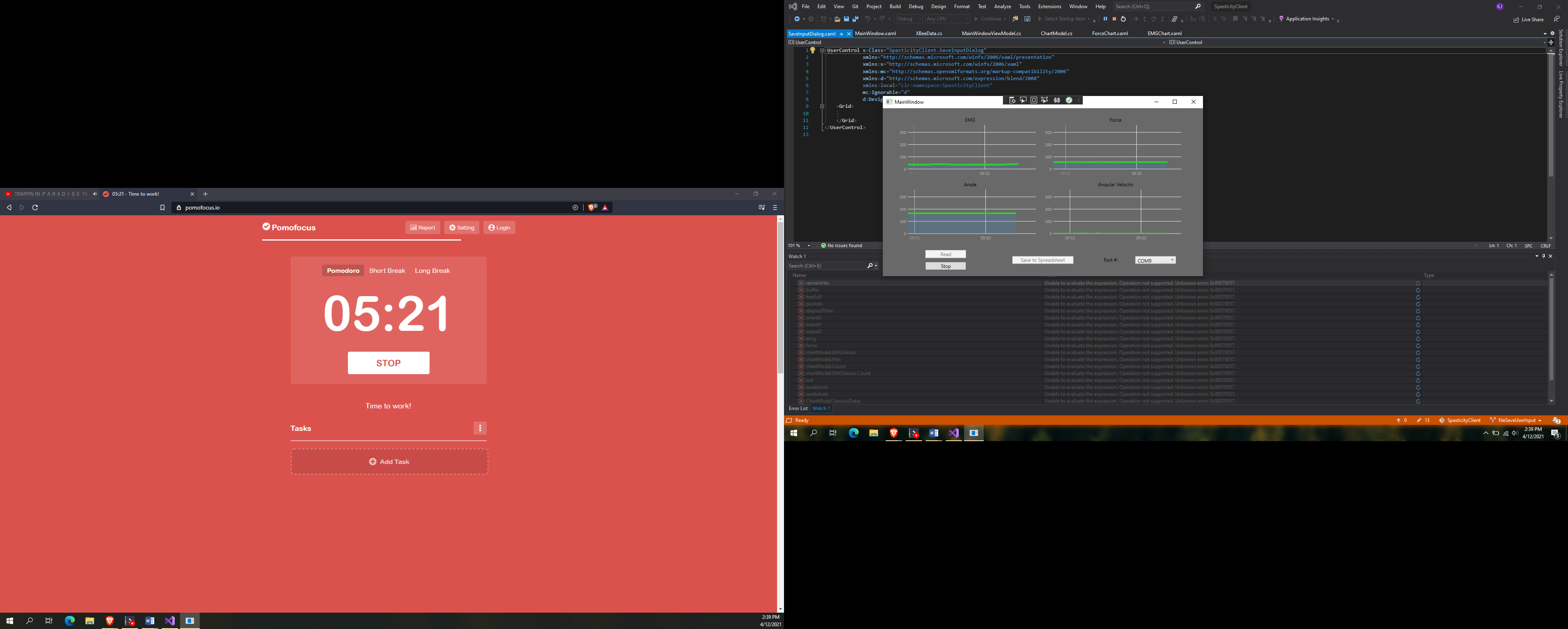
SpasticityClient Development Guide

# Program Introduction

SpasticityClient is a prototype application used as a diagnostic aid for muscle spasticity. It is built on Windows Presentation Foundation, a UI framework that creates desktop client applications. The framework is part of .NET and uses Extensible Application Markup Language (XAML) for the User Interface (UI) and C# for back end code. The goal of this guide is to document the architecture of the program and to make it easy to do further development.

The application receives data from an xBee receiver which connects to the computer over USB. The receiver listens for data from a device consisting of two inertial measurement units (IMUs), a capacitive force sensor, and an EMG sensor, which is worn by the individual undergoing assessment. The output of this instrument is Force in N, EMG in mV, joint angle in degrees, and joint angular velocity in degrees per second.

The program currently has capability to open the serial port, plot the force, EMG, and joint angle values, stop the session and record the data to an Excel sheet. Below is an image of the program interface.



# Development Environment and Version Management

Version management for this project is done on Github. Any way of using Git (e.g. command line) will achieve the same purpose but in this case Github Desktop was used. The production branch is called main. This is the branch that will contain the code for the final product. The main development branch is called dev. When a new feature is added, a feature branch is created and tested until it is working, after which it is merged with the dev branch. The repository can be found and cloned at https://github.com/khilesh55/SpasticityClient.

Visual Studio 2019 was used for editing the code. Several features of Visual Studio are useful for the development of this project: Intellicode for auto-completing code and providing suggestions, the debugger feature, and the object browser among others. Visual Studio can be downloaded at https://visualstudio.microsoft.com/downloads/.

# Program Pattern

The program is built based on the Model-View-ViewModel (MVVM) pattern. This pattern is beneficial due to advantages in development speed, clarity of code among other benefits. The accompanying slideshow contains more information about this pattern. To summarize, the components of the pattern are:

*Model* – This is the data object.

*View* – What the user sees on their screen and allows the user to interact with back end logic. Binds to properties on viewmodel.

*ViewModel* – Code files with properties and functions needed for the views to operate. Expose data objects from the model.

The code files themselves are commented in detail. The functions of the key code files are summarized for convenience in the table below.

|  |  |  |
| --- | --- | --- |
| **Code File** | **MVVM Role** | **Description** |
| AngleChart.xaml | View (Child) | Graph display for Angle |
| AngularVelocityChart.xaml | View (Child) | Graph display for Angular Velocity |
| EMGChart.xaml | View (Child) | Graph display for EMG |
| ForceChart.xaml | View (Child) | Graph display for Force |
| MainWindow.xaml | View (Parent) | Main window display with: 4 graph views for each measurement, start and stop buttons, and port selector dropdown |
| ChartModel.cs | ViewModel (Child) | The chart model contains: data arrays for each chart type, a list of session data (SessionDatas), implements command interface for read, stop and save; min and max properties for axis formatting and a function to dynamically set axis limits. Constructor takes a port name supplied by the dropdown and initializes a new XBeeData for the session. Also includes excel save method. Read method invokes read method from XBeeData. |
| MainWindowViewModel.cs | ViewModel (Parent) | Main window view model contains a list of portnames bound to the combobox on the main window, instantiates a ChartModel object for the session; registers the trial license for SyncFusion |
| RelayCommand.cs | Command Interface | Contains ApplicationCommands class which uses the Prism library. RelayCommand can probably be removed |
| SessionData.cs | Model | Class with properties corresponding to each measured quantity (e.g. EMG, Force, etc as well as a timestamp. This is so that a list of these objects can be stored in the ChartModel |
| XBeeData.cs | Model | Contains function to read from serial port, convert values from hex to long, send data to ChartModel (chart value collection), and send data to Excel (SessionDatas) |
| XBeeFunctions.cs | Model | Contains class defining one transmitted XBee packet, and functions to separate incoming data into relevant portions (header, data, etc) |

# External Dependencies

Some external libraries are used in the application. If Visual Studio is used as the development environment, these packages are easy to install using the NuGet package manager. Details on the NuGet package manager can be found at <https://docs.microsoft.com/en-us/nuget/consume-packages/install-use-packages-visual-studio>. Alternatively, one can download the packages at nuget.org. The dependencies that were used are detailed in the table below.

|  |  |  |
| --- | --- | --- |
| **Dependency** | **Documentation** | **Description** |
| Syncfusion.Shared.WPF | https://www.syncfusion.com/wpf-controls | Charting library not currently implemented in program |
| Syncfusion.XlsIO.WPF | https://help.syncfusion.com/file-formats/xlsio/overview | Makes Excel manipulation possible from within the application |
| Syncfusion.Compression.Base | https://help.syncfusion.com/file-formats/xlsio/overview | Helper library needed for XlsIO |
| Syncfusion.Licensing | https://help.syncfusion.com/common/essential-studio/licensing/license-key | Syncfusion licensing |
| LiveCharts | https://lvcharts.net/App/examples/v1/wpf/Install | Charting library currently being used |
| LiveCharts.WPF | https://lvcharts.net/App/examples/v1/wpf/Install | Both LiveCharts and LiveCharts.WPF are needed for charting. This contains the charting controls |
| Prism.WPF | https://prismlibrary.com/docs/getting-started/Download-and-Setup-Prism.html | For setting up the commanding interface, enables one button to issue multiple commands |
| Prism.Core | https://prismlibrary.com/docs/getting-started/Download-and-Setup-Prism.html | Both Prism.WPF and Prism.Core are needed for commanding interface |

# Roadmap

There are still some important updates that the project will benefit from. Below is a roadmap for upcoming work on the project.

Apr 19 – 22: Research IMU absolute angle measurement & convert to code, verify Excel file save feature working, work on solid model v1.

Apr 22: Project update and code review, enclosure v1 showcase

# Joint Angle Measurement

The joint angle can be found by accessing the quaternion output from each IMU and applying the following formula. https://lup.lub.lu.se/luur/download?func=downloadFile&recordOId=8996696&fileOId=8996707

Theta = 2\*arccos(|qw1\*qw2 + qx1\*qx2 + qy1\*qy2+ qz1\*qz2|)

According to the study both absolute and relative modes may be used, though relative mode is more accurate. The relative mode requires the devices to be turned on while sharing the same orientation with one another.

