Benjamin Kuo
Team C: Mell-E
Teammates: Yoga Y Nadaraaan, Michael Vander Meiden, Nima Rahnemoon, Kai Li
ILR01
October 14, 2016

1. Individual Progress - For the sensor and motorlab, I was focused on the following 3 components
    a. System Integration / Software Architecture Design - The first thing for this lab I wanted to set up, was an architecture for easy integration. I requested from all of the members from our team to complete a C++ Class/API for each type of component they wanted to work on. For each sensor class there was generally some sort of initialization function/constructor and a member function to request the current a sensor reading. As for the actuator classes, the same follows with a initialization function/constructor, but instead of the reading function, it generally was a way to send some sort of control effort command. With these general functions, I built a general state machine that would accept a sensor selection and an actuator selection that would be pulled from the GUI and map the sensor value to an actuator control effort value. A significant part of this project also revolved around ROS too. This wasn't required, but we thought it would be a good idea to familiarize ourselves with ROS a bit more because we will likely use ROS to control our robot. Because the amount of data we need to process it seemed it would be inefficient to create a dozen publishers and subscribers, so I started out by creating two custom messages with primitive data types. One was the GUI information to send to the Arduino and one was for the Arduino to transfer information back to the PC. The one problem with this setup however is that the Arduino doesn't process floating point numbers very well, so we decided to pass only raw ADC values back to the PC. Because of the limitations of the GUI, it wasn't possible to have conversion factors inside the data logging, and I had to write small node for ROS to take the raw ADC values and convert those to normal units to report back to the GUI interface. In doing this I had to create another custom message for a set of "translated" values.
    b. GUI - Since I had limited experience with GUI's, but I also wanted to learn a bit more about GUI's I took it upon myself to look into solutions. Since we were using ROS I discovered that ROS actually has rqt as tool for doing robot visualizations, and the framework allows you to drag and drop various plugins into the rqt interface. The plotting feature was the backbone of the way we were reporting our sensor data, and I had just dragged and dropped 5 different plotting plugins for this purpose. For our binary data, this didn't make as much sense so I simply used the rqt topic list plugin to show the boolean values of those sensors. The built-in plugins were really easy to use for showing data, but publishing data to

the Arduino was more work, because we had to write a custom plugin to send our data. Given the limited amount of time, and the fact that these rqt plugins were largely open source, I was able to heavily modify the robot steering plugin and convert it to our needs. This plugin featured the the text fields and buttons that we needed for our GUI interface but lacked the checkboxes, so I had to find another rqt plugin that had these and copy some of the code from that. As mentioned earlier this part of the GUI sends a mapping of sensor to actuator control effort.

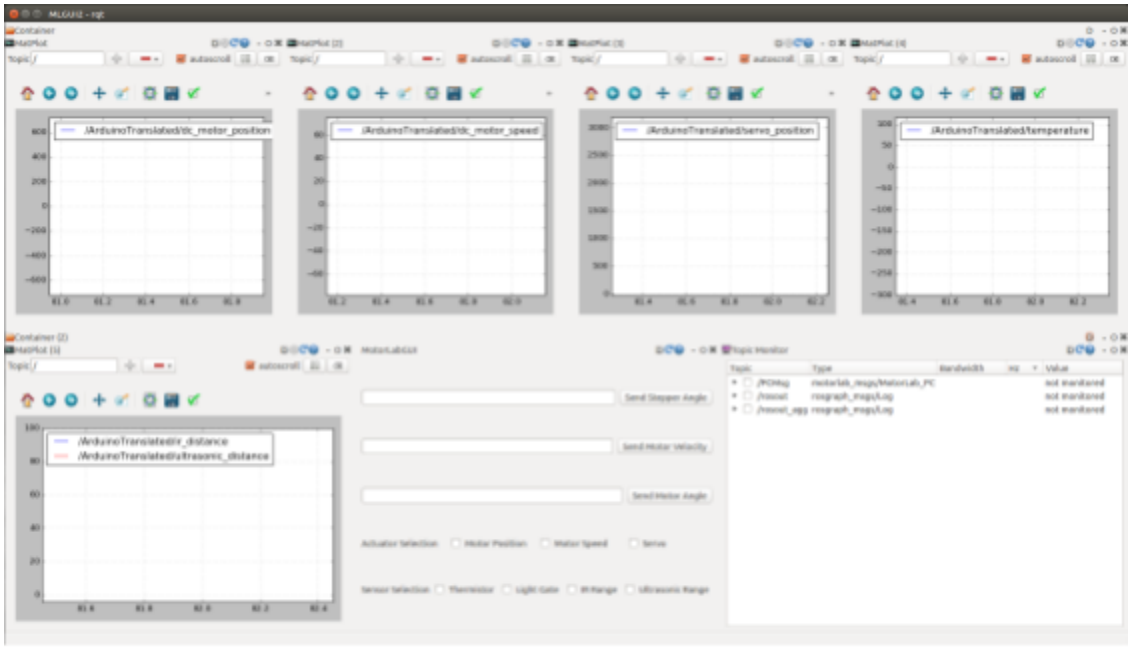

Figure 1: Our rqt GUI

c. Support for other team members - In addition to the work that I had done for the GUI and the software integration. I was also answering questions and helping my teammates understand other aspects of this assignment. I was helping Nima understand the theory behind PID, and I was looking through his code to debug. The light gate sensor also wasn't behaving as expected, so when Kai was trying to set up the sensor I was using my electrical background to give suggestions to Kai on where to fix the hardware setup.

2. Challenges
   a. ROS Custom Messages - For the custom message creation. The act of making them wasn't terribly difficult, but the steps involved to convert them into .h files for Arduino took much more time than expected. The fix for the solution was not difficult, but the hard part was figuring out why the setting up of the Arduino libraries were not working. Basically when trying to

generate the Arduino ROS message headers, many refused to be created. I think there was simply a problem with the permissions of the folder that I was trying to create the messages in because I was able to resolve this my importing the files into a temporary directory then copying them manually to the Arduino library folder.

   b. rqt - When we started using this we were very concerned why dragging and dropping the different plugins would cause rqt to crash. Rqt inherently has some problems when dragging and dropping, but I discovered that if you created “containers” within the rqt environment, the system was much less likely to crash when dragging and dropping plugins within the “containers”. I also didn’t have enough time to formally get familar with Qt, so I had to hack things together really quickly. This resulted in a limited understanding of Qt, but miraculously the GUI worked.

3. Teamwork - Since one of my main roles was software integration I had to interface will all of the other members. As mentioned above I laid out my expectations for each of the C++ Classes/APIs, and I was able to help team members understand some of the problems they were facing (PID with Nima, and the Light Gate with Kai). Finally, Yoga talked to me about putting some final touches on the messages and the GUI, and he carried the final touches out. As far as the team members’ individual work. Nima’s focus was on the DC motor, Kai’s was on the Thermistor and the Light Gate hardware setup. Michael worked largely on the stepper motor in both code and hardware. Yoga focused on the integration of the Servo, Range Sensors and the Code for the button and Lightgate.
4. Plans - Though my main responsibility is the manipulation of the litter, none of those materials are in our possession yet so I can’t really work on that. Instead I will try to help Michael on the obstacle detection task. He will try to do some data acquisition from the Velodyne Puck, and I will try to extract data from the Hokuyo. We will try out the various ROS API’s available, and try to figure out if the performances are desired, the API’s are viable for quick integration within our project, and if the infrastructure for the sensor is overly complicated. The ultimate goal is to decide on one of these 2 sensors to be our main obstacle detection sensor.

Code I Contributed To

MotorLab_Arduino.msg

```
int16 dc_motor_position
int16 dc_motor_speed
int16 servo_position
int16 stepper_motor_position
int16 temperature
int16 light_gate_state
int16 ultrasonic_distance
int16 ir_distance
int16 button_state
```

MotorLab_Arduino_Translation.msg

```
float32 dc_motor_position
float32 dc_motor_speed
float32 servo_position
float32 stepper_motor_position
float32 temperature
float32 light_gate_state
float32 ultrasonic_distance
float32 ir_distance
float32 button_state
```

MotorLab_PC.msg

```
int16 stepper_angle
int16 motor_position_checked
int16 motor_speed_checked
int16 servo_checked
int16 thermistor_checked
int16 light_gate_checked
int16 ir_checked
int16 ultra_checked
int16 motor_velocity
int16 motor_angle
```

MotorlabGUI.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
```

```
<class>MotorlabGUIWidget</class>
<widget class="QWidget" name="MotorlabGUIWidget">
  <property name="geometry">
    <rect>
      <x>0</x>
      <y>0</y>
      <width>400</width>
      <height>344</height>
    </rect>
  </property>
  <property name="windowTitle">
    <string>MotorLabGUI</string>
  </property>
  <layout class="QVBoxLayout" name="verticalLayout" stretch="0,0,0,0,0">
    <item>
      <layout class="QHBoxLayout" name="horizontalLayout">
        <item>
          <widget class="QLineEdit" name="set_stepper">
            <property name="toolTip">
              <string>Stepper Angle</string>
            </property>
            <property name="text">
              <string/>
            </property>
          </widget>
        </item>
        <item>
          <widget class="QPushButton" name="send_stepper">
            <property name="toolTip">
              <string>Send Stepper Angle</string>
            </property>
            <property name="text">
              <string>Send Stepper Angle</string>
            </property>
          </widget>
        </item>
      </layout>
    </item>
    <item>
```

```
 <layout class="QHBoxLayout" name="horizontalLayout">
  <item>
 <layout class="QHBoxLayout" name="horizontalLayout">
  <item>
   <widget class="QLineEdit" name="set_motor_velocity">
    <property name="toolTip">
     <string>Motor velocity</string>
    </property>
    <property name="text">
     <string/>
    </property>
   </widget>
  </item>
  <item>
   <widget class="QPushButton" name="send_motor_velocity">
    <property name="toolTip">
     <string>Send Motor Velocity</string>
    </property>
    <property name="text">
     <string>Send Motor Velocity</string>
    </property>
   </widget>
  </item>
 </layout>
</item>
 </layout>
</item>
<item>
 <layout class="QHBoxLayout" name="horizontalLayout">
  <item>
 <layout class="QHBoxLayout" name="horizontalLayout">
  <item>
   <widget class="QLineEdit" name="set_motor_angle">
    <property name="toolTip">
     <string>Motor angle</string>
    </property>
    <property name="text">
     <string/>
    </property>
```

```
      </widget>
    </item>
    <item>
      <widget class="QPushButton" name="send_motor_angle">
        <property name="toolTip">
          <string>Send Motor Angle</string>
        </property>
        <property name="text">
          <string>Send Motor Angle</string>
        </property>
      </widget>
    </item>
  </layout>
</item>
</layout>
</item>
<item>
  <layout class="QHBoxLayout" name="horizontalLayout">
    <item>
      <widget class="QLabel" name="Actuator_Selection">
        <property name="toolTip">
          <string>Actuator Selection</string>
        </property>
        <property name="text">
          <string>Actuator Selection</string>
        </property>
      </widget>
    </item>
    <!-- <item> -->
      <!-- <widget class="QButtonGroup" name="MotorButtonGroup"> -->
        <!-- <property name="exclusive"> -->
          <!-- <bool>true</bool> -->
        <!-- </property> -->
        <item>
          <widget class="QCheckBox" name="motor_position_checkbox">
            <property name="text">
              <string>Motor Position</string>
            </property>
            <property name="checked">
```

```
              <bool>false</bool>
             </property>
            </widget>
           </item>
           <item>
            <widget class="QCheckBox" name="motor_speed_checkbox">
             <property name="text">
              <string>Motor Speed</string>
             </property>
             <property name="checked">
              <bool>false</bool>
             </property>
            </widget>
           </item>
          <!-- </widget> -->
         <!-- </item> -->
         <item>
          <widget class="QCheckBox" name="servo_checkbox">
           <property name="text">
            <string>Servo</string>
           </property>
           <property name="checked">
            <bool>false</bool>
           </property>
          </widget>
         </item>
<!--          <item>
          <widget class="QCheckBox" name="autoscroll_checkbox">
           <property name="text">
            <string>autoscroll</string>
           </property>
           <property name="checked">
            <bool>false</bool>
           </property>
          </widget>
         </item> -->
        </layout>
       </item>
       <item>
```

```
<layout class="QHBoxLayout" name="horizontalLayout">
 <!-- <item> -->
  <!-- <widget class="QButtonGroup" name="SensorButtonGroup"> -->
   <!-- <property name="exclusive"> -->
    <!-- <bool>true</bool> -->
   <!-- </property> -->
  <item>
   <widget class="QLabel" name="Sensor_Selection">
    <property name="toolTip">
     <string>Sensor Selection</string>
    </property>
    <property name="text">
     <string>Sensor Selection</string>
    </property>
   </widget>
  </item>
  <item>
   <widget class="QCheckBox" name="thermistor_checkbox">
    <property name="text">
     <string>Thermistor</string>
    </property>
    <property name="checked">
     <bool>false</bool>
    </property>
   </widget>
  </item>
  <item>
   <widget class="QCheckBox" name="light_gate_checkbox">
    <property name="text">
     <string>Light Gate</string>
    </property>
    <property name="checked">
     <bool>false</bool>
    </property>
   </widget>
  </item>
  <item>
   <widget class="QCheckBox" name="ir_checkbox">
    <property name="text">
```

```
              <string>IR Range</string>
             </property>
             <property name="checked">
              <bool>false</bool>
             </property>
            </widget>
           </item>
           <item>
            <widget class="QCheckBox" name="ultra_checkbox">
             <property name="text">
              <string>Ultrasonic Range</string>
             </property>
             <property name="checked">
              <bool>false</bool>
             </property>
            </widget>
           </item>
          <!-- </widget> -->
        <!-- </item> -->
      </layout>
    </item>
<!--      <item>
       <layout class="QHBoxLayout" name="horizontalLayout">
        <item>
         <widget class="QLabel" name="Binary_Sensor_States">
          <property name="toolTip">
           <string>Binary_Sensor_States</string>
          </property>
          <property name="text">
           <string>Binary_Sensor_States</string>
          </property>
         </widget>
        </item>
        <item>
         <widget class="QLabel" name="Light_Gate_State">
          <property name="toolTip">
           <string>Light Gate State</string>
          </property>
          <property name="text">
```

```
            <string>Light Gate State: 0</string>
           </property>
          </widget>
         </item>
         <item>
          <widget class="QLabel" name="Button_State">
           <property name="toolTip">
            <string>Button State</string>
           </property>
           <property name="text">
            <string>Actuators: Off</string>
           </property>
          </widget>
         </item>
        </layout>
       </item> -->
       <!-- <item>
       <widget class="QCheckBox" name="autoscroll_checkbox">
        <property name="text">
         <string>autoscroll</string>
        </property>
        <property name="checked">
         <bool>true</bool>
        </property>
       </widget>
      </item> -->
<!--       <item>
       <layout class="QHBoxLayout" name="horizontalLayout">
        <item>
         <widget class="QLineEdit" name="topic_line_edit">
          <property name="toolTip">
           <string>Topic to send Twist message to</string>
          </property>
          <property name="text">
           <string/>
          </property>
         </widget>
        </item>
        <item>
```

```
     <widget class="QPushButton" name="stop_push_button">
      <property name="toolTip">
       <string>Reset velocities to zero</string>
      </property>
      <property name="text">
       <string>Stop</string>
      </property>
     </widget>
    </item>
   </layout>
  </item>
  <item>
   <layout class="QHBoxLayout" name="horizontalLayout_5" stretch="1,0,0,0,1">
    <item>
     <spacer name="horizontalSpacer">
      <property name="orientation">
       <enum>Qt::Horizontal</enum>
      </property>
      <property name="sizeHint" stdset="0">
       <size>
        <width>40</width>
        <height>20</height>
       </size>
      </property>
     </spacer>
    </item>
    <item>
     <layout class="QVBoxLayout" name="verticalLayout_2">
      <property name="leftMargin">
       <number>40</number>
      </property>
      <item>
       <widget class="QPushButton" name="increase_x_linear_push_button">
        <property name="maximumSize">
         <size>
          <width>30</width>
          <height>16777215</height>
         </size>
        </property>
```

```
    <property name="toolTip">
     <string>Increase linear  velocity</string>
    </property>
    <property name="text">
     <string>+</string>
    </property>
   </widget>
  </item>
  <item>
   <spacer name="verticalSpacer">
    <property name="orientation">
     <enum>Qt::Vertical</enum>
    </property>
    <property name="sizeHint" stdset="0">
     <size>
      <width>20</width>
      <height>40</height>
     </size>
    </property>
   </spacer>
  </item>
  <item>
   <widget class="QPushButton" name="reset_x_linear_push_button">
    <property name="maximumSize">
     <size>
      <width>30</width>
      <height>16777215</height>
     </size>
    </property>
    <property name="toolTip">
     <string>Reset linear  velocity</string>
    </property>
    <property name="text">
     <string>0</string>
    </property>
   </widget>
  </item>
  <item>
   <spacer name="verticalSpacer_2">
```

```
        <property name="orientation">
         <enum>Qt::Vertical</enum>
        </property>
        <property name="sizeHint" stdset="0">
         <size>
          <width>20</width>
          <height>40</height>
         </size>
        </property>
       </spacer>
      </item>
      <item>
       <widget class="QPushButton" name="decrease_x_linear_push_button">
        <property name="maximumSize">
         <size>
          <width>30</width>
          <height>16777215</height>
         </size>
        </property>
        <property name="toolTip">
         <string>Decrease linear  velocity</string>
        </property>
        <property name="text">
         <string>-</string>
        </property>
       </widget>
      </item>
     </layout>
    </item>
    <item>
     <widget class="QSlider" name="x_linear_slider">
      <property name="minimum">
       <number>-1000</number>
      </property>
      <property name="maximum">
       <number>1000</number>
      </property>
      <property name="singleStep">
       <number>10</number>
```

```
   </property>
   <property name="pageStep">
    <number>100</number>
   </property>
   <property name="orientation">
    <enum>Qt::Vertical</enum>
   </property>
   <property name="tickPosition">
    <enum>QSlider::TicksBothSides</enum>
   </property>
  </widget>
 </item>
 <item>
  <layout class="QVBoxLayout" name="verticalLayout_3" stretch="0,0,0,0,0">
   <item>
    <widget class="QDoubleSpinBox" name="max_x_linear_double_spin_box">
     <property name="toolTip">
      <string>Maximum linear  velocity</string>
     </property>
     <property name="maximum">
      <double>10.000000000000000</double>
     </property>
     <property name="singleStep">
      <double>0.100000000000000</double>
     </property>
     <property name="value">
      <double>1.000000000000000</double>
     </property>
    </widget>
   </item>
   <item>
    <spacer name="verticalSpacer_3">
     <property name="orientation">
      <enum>Qt::Vertical</enum>
     </property>
     <property name="sizeHint" stdset="0">
      <size>
       <width>20</width>
       <height>40</height>
```

```
      </size>
     </property>
    </spacer>
   </item>
   <item>
    <widget class="QLabel" name="current_x_linear_label">
     <property name="toolTip">
      <string>Current linear velocity</string>
     </property>
     <property name="text">
      <string>0.0 m/s</string>
     </property>
    </widget>
   </item>
   <item>
    <spacer name="verticalSpacer_4">
     <property name="orientation">
      <enum>Qt::Vertical</enum>
     </property>
     <property name="sizeHint" stdset="0">
      <size>
       <width>20</width>
       <height>40</height>
      </size>
     </property>
    </spacer>
   </item>
   <item>
    <widget class="QDoubleSpinBox" name="min_x_linear_double_spin_box">
     <property name="toolTip">
      <string>Minimum linear  velocity</string>
     </property>
     <property name="minimum">
      <double>-10.000000000000000</double>
     </property>
     <property name="maximum">
      <double>0.000000000000000</double>
     </property>
     <property name="singleStep">
```

```
            <double>0.100000000000000</double>
           </property>
           <property name="value">
            <double>-1.000000000000000</double>
           </property>
          </widget>
         </item>
        </layout>
       </item>
       <item>
        <spacer name="horizontalSpacer_2">
         <property name="orientation">
          <enum>Qt::Horizontal</enum>
         </property>
         <property name="sizeHint" stdset="0">
          <size>
           <width>40</width>
           <height>20</height>
          </size>
         </property>
        </spacer>
       </item>
      </layout>
     </item>
     <item>
      <widget class="QSlider" name="z_angular_slider">
       <property name="minimum">
        <number>-3000</number>
       </property>
       <property name="maximum">
        <number>3000</number>
       </property>
       <property name="singleStep">
        <number>10</number>
       </property>
       <property name="pageStep">
        <number>100</number>
       </property>
       <property name="orientation">
```

```
      <enum>Qt::Horizontal</enum>
     </property>
     <property name="invertedAppearance">
      <bool>true</bool>
     </property>
     <property name="tickPosition">
      <enum>QSlider::TicksBelow</enum>
     </property>
    </widget>
   </item>
   <item>
    <layout class="QHBoxLayout" name="horizontalLayout_3">
     <item>
      <widget class="QPushButton" name="increase_z_angular_push_button">
       <property name="maximumSize">
        <size>
         <width>30</width>
         <height>16777215</height>
        </size>
       </property>
       <property name="toolTip">
        <string>Increase angular velocity</string>
       </property>
       <property name="text">
        <string>&amp;lt;</string>
       </property>
      </widget>
     </item>
     <item>
      <spacer name="horizontalSpacer_3">
       <property name="orientation">
        <enum>Qt::Horizontal</enum>
       </property>
       <property name="sizeHint" stdset="0">
        <size>
         <width>40</width>
         <height>20</height>
        </size>
       </property>
```

```
  </spacer>
</item>
<item>
  <widget class="QPushButton" name="reset_z_angular_push_button">
    <property name="maximumSize">
      <size>
        <width>30</width>
        <height>16777215</height>
      </size>
    </property>
    <property name="toolTip">
      <string>Reset angular velocity</string>
    </property>
    <property name="text">
      <string>0</string>
    </property>
  </widget>
</item>
<item>
  <spacer name="horizontalSpacer_4">
    <property name="orientation">
      <enum>Qt::Horizontal</enum>
    </property>
    <property name="sizeHint" stdset="0">
      <size>
        <width>40</width>
        <height>20</height>
      </size>
    </property>
  </spacer>
</item>
<item>
  <widget class="QPushButton" name="decrease_z_angular_push_button">
    <property name="maximumSize">
      <size>
        <width>30</width>
        <height>16777215</height>
      </size>
    </property>
```

```
        <property name="toolTip">
         <string>Decrease angular velocity</string>
        </property>
        <property name="text">
         <string>&gt;</string>
        </property>
       </widget>
      </item>
     </layout>
    </item>
    <item>
     <layout class="QHBoxLayout" name="horizontalLayout_4" stretch="0,0,0,0,0">
      <item>
       <widget class="QDoubleSpinBox" name="max_z_angular_double_spin_box">
        <property name="toolTip">
         <string>Maximum angular velocity</string>
        </property>
        <property name="maximum">
         <double>10.000000000000000</double>
        </property>
        <property name="singleStep">
         <double>0.100000000000000</double>
        </property>
        <property name="value">
         <double>3.000000000000000</double>
        </property>
       </widget>
      </item>
      <item>
       <spacer name="horizontalSpacer_5">
        <property name="orientation">
         <enum>Qt::Horizontal</enum>
        </property>
        <property name="sizeHint" stdset="0">
         <size>
          <width>40</width>
          <height>20</height>
         </size>
        </property>
```

```
  </spacer>
 </item>
 <item>
  <widget class="QLabel" name="current_z_angular_label">
   <property name="toolTip">
    <string>Current angular velocity</string>
   </property>
   <property name="text">
    <string>0.0 rad/s</string>
   </property>
  </widget>
 </item>
 <item>
  <spacer name="horizontalSpacer_6">
   <property name="orientation">
    <enum>Qt::Horizontal</enum>
   </property>
   <property name="sizeHint" stdset="0">
    <size>
     <width>40</width>
     <height>20</height>
    </size>
   </property>
  </spacer>
 </item>
 <item>
  <widget class="QDoubleSpinBox" name="min_z_angular_double_spin_box">
   <property name="toolTip">
    <string>Minimum angular velocity</string>
   </property>
   <property name="minimum">
    <double>-10.000000000000000</double>
   </property>
   <property name="maximum">
    <double>0.000000000000000</double>
   </property>
   <property name="singleStep">
    <double>0.100000000000000</double>
   </property>
```

```
            <property name="value">
              <double>-3.000000000000000</double>
            </property>
           </widget>
          </item>
         </layout>
        </item> -->
  </layout>
</widget>
<resources/>
<connections/>
</ui>
```

motorlab_gui.py

```
# Copyright (c) 2011, Dirk Thomas, TU Darmstadt
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without
# modification, are permitted provided that the following conditions
# are met:
#
#   * Redistributions of source code must retain the above copyright
#     notice, this list of conditions and the following disclaimer.
#   * Redistributions in binary form must reproduce the above
#     copyright notice, this list of conditions and the following
#     disclaimer in the documentation and/or other materials provided
#     with the distribution.
#   * Neither the name of the TU Darmstadt nor the names of its
#     contributors may be used to endorse or promote products derived
#     from this software without specific prior written permission.
#
# THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS
# "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
# LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
# FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
# COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
INDIRECT,
```

```
# INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING,
# BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES;
# LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
# CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT
# LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
# ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
# POSSIBILITY OF SUCH DAMAGE.

from __future__ import division
import os
import rospkg

from geometry_msgs.msg import Twist
import rospy
from python_qt_binding import loadUi
from python_qt_binding.QtCore import Qt, QTimer, Slot
from python_qt_binding.QtGui import QKeySequence
from python_qt_binding.QtWidgets import QShortcut, QWidget
from rqt_gui_py.plugin import Plugin

# from motorlab_msgs.msg import MotorLab_Arduino
# from motorlab_msgs.msg import MotorLab_Arduino_Translation
from motorlab_msgs.msg import MotorLab_PC

class MotorlabGUI(Plugin):

    slider_factor = 1000.0

    def __init__(self, context):
        super(MotorlabGUI, self).__init__(context)
        self.setObjectName('MotorlabGUI')

        # self._publisher = None
        self._publisher = rospy.Publisher("PCMsg", MotorLab_PC, queue_size = 2)

        self._widget = QWidget()
```

```
        rp = rospkg.RosPack()
        ui_file = os.path.join(rp.get_path('rqt_motorlab_gui'), 'resource', 'MotorlabGUI.ui')
        loadUi(ui_file, self._widget)
        self._widget.setObjectName('MotorlabGUIUi')
        if context.serial_number() > 1:
            self._widget.setWindowTitle(self._widget.windowTitle() + (' (%d)' %
context.serial_number()))
        context.add_widget(self._widget)

        self.PC_to_Arduino = MotorLab_PC()

        self.PC_to_Arduino.stepper_angle = 0

        self.PC_to_Arduino.motor_position_checked = 0
        self.PC_to_Arduino.motor_speed_checked = 0
        self.PC_to_Arduino.servo_checked = 0
        self.PC_to_Arduino.thermistor_checked = 0
        self.PC_to_Arduino.light_gate_checked = 0
        self.PC_to_Arduino.ir_checked = 0
        self.PC_to_Arduino.ultra_checked = 0
        self.PC_to_Arduino.motor_velocity = 0
        self.PC_to_Arduino.motor_angle = 0

        self.text_field_angle = 0
        self.text_field_motor_velocity= 0
        self.text_field_motor_angle = 0

        self._widget.set_stepper.textChanged.connect(self._on_set_stepper_changed)
        self._widget.send_stepper.pressed.connect(self._on_send_stepper_pressed)

        self._widget.set_motor_velocity.textChanged.connect(self._on_set_motor_velocity)
        self._widget.send_motor_velocity.pressed.connect(self._on_send_motor_velocity)

        self._widget.set_motor_angle.textChanged.connect(self._on_set_motor_angle)
        self._widget.send_motor_angle.pressed.connect(self._on_send_motor_angle)


self._widget.motor_position_checkbox.clicked.connect(self._on_motor_position_checkb
ox_clicked)
```

```
self._widget.motor_speed_checkbox.clicked.connect(self._on_motor_speed_checkbox_clicked)
        self._widget.servo_checkbox.clicked.connect(self._on_servo_checkbox_clicked)

self._widget.thermistor_checkbox.clicked.connect(self._on_thermistor_checkbox_clicked)

self._widget.light_gate_checkbox.clicked.connect(self._on_light_gate_checkbox_clicked)
        self._widget.ir_checkbox.clicked.connect(self._on_ir_checkbox_clicked)
        self._widget.ultra_checkbox.clicked.connect(self._on_ultra_checkbox_clicked)


        # self._widget.topic_line_edit.textChanged.connect(self._on_topic_changed)
        # self._widget.stop_push_button.pressed.connect(self._on_stop_pressed)

        #
self._widget.x_linear_slider.valueChanged.connect(self._on_x_linear_slider_changed)
        #
self._widget.z_angular_slider.valueChanged.connect(self._on_z_angular_slider_changed)

        #
self._widget.increase_x_linear_push_button.pressed.connect(self._on_strong_increase_x_linear_pressed)
        #
self._widget.reset_x_linear_push_button.pressed.connect(self._on_reset_x_linear_pressed)
        #
self._widget.decrease_x_linear_push_button.pressed.connect(self._on_strong_decrease_x_linear_pressed)
        #
self._widget.increase_z_angular_push_button.pressed.connect(self._on_strong_increase_z_angular_pressed)
        #
self._widget.reset_z_angular_push_button.pressed.connect(self._on_reset_z_angular_pressed)
```

```
        #
self._widget.decrease_z_angular_push_button.pressed.connect(self._on_strong_decre
ase_z_angular_pressed)

        #
self._widget.max_x_linear_double_spin_box.valueChanged.connect(self._on_max_x_li
near_changed)
        #
self._widget.min_x_linear_double_spin_box.valueChanged.connect(self._on_min_x_lin
ear_changed)
        #
self._widget.max_z_angular_double_spin_box.valueChanged.connect(self._on_max_z_
angular_changed)
        #
self._widget.min_z_angular_double_spin_box.valueChanged.connect(self._on_min_z_a
ngular_changed)

        # self.shortcut_w = QShortcut(QKeySequence(Qt.Key_W), self._widget)
        # self.shortcut_w.setContext(Qt.ApplicationShortcut)
        # self.shortcut_w.activated.connect(self._on_increase_x_linear_pressed)
        # self.shortcut_x = QShortcut(QKeySequence(Qt.Key_X), self._widget)
        # self.shortcut_x.setContext(Qt.ApplicationShortcut)
        # self.shortcut_x.activated.connect(self._on_reset_x_linear_pressed)
        # self.shortcut_s = QShortcut(QKeySequence(Qt.Key_S), self._widget)
        # self.shortcut_s.setContext(Qt.ApplicationShortcut)
        # self.shortcut_s.activated.connect(self._on_decrease_x_linear_pressed)
        # self.shortcut_a = QShortcut(QKeySequence(Qt.Key_A), self._widget)
        # self.shortcut_a.setContext(Qt.ApplicationShortcut)
        # self.shortcut_a.activated.connect(self._on_increase_z_angular_pressed)
        # self.shortcut_z = QShortcut(QKeySequence(Qt.Key_Z), self._widget)
        # self.shortcut_z.setContext(Qt.ApplicationShortcut)
        # self.shortcut_z.activated.connect(self._on_reset_z_angular_pressed)
        # self.shortcut_d = QShortcut(QKeySequence(Qt.Key_D), self._widget)
        # self.shortcut_d.setContext(Qt.ApplicationShortcut)
        # self.shortcut_d.activated.connect(self._on_decrease_z_angular_pressed)

        # self.shortcut_shift_w = QShortcut(QKeySequence(Qt.SHIFT + Qt.Key_W),
self._widget)
        # self.shortcut_shift_w.setContext(Qt.ApplicationShortcut)
```

```
        #
self.shortcut_shift_w.activated.connect(self._on_strong_increase_x_linear_pressed)
        # self.shortcut_shift_x = QShortcut(QKeySequence(Qt.SHIFT + Qt.Key_X),
self._widget)
        # self.shortcut_shift_x.setContext(Qt.ApplicationShortcut)
        # self.shortcut_shift_x.activated.connect(self._on_reset_x_linear_pressed)
        # self.shortcut_shift_s = QShortcut(QKeySequence(Qt.SHIFT + Qt.Key_S),
self._widget)
        # self.shortcut_shift_s.setContext(Qt.ApplicationShortcut)
        #
self.shortcut_shift_s.activated.connect(self._on_strong_decrease_x_linear_pressed)
        # self.shortcut_shift_a = QShortcut(QKeySequence(Qt.SHIFT + Qt.Key_A),
self._widget)
        # self.shortcut_shift_a.setContext(Qt.ApplicationShortcut)
        #
self.shortcut_shift_a.activated.connect(self._on_strong_increase_z_angular_pressed)
        # self.shortcut_shift_z = QShortcut(QKeySequence(Qt.SHIFT + Qt.Key_Z),
self._widget)
        # self.shortcut_shift_z.setContext(Qt.ApplicationShortcut)
        # self.shortcut_shift_z.activated.connect(self._on_reset_z_angular_pressed)
        # self.shortcut_shift_d = QShortcut(QKeySequence(Qt.SHIFT + Qt.Key_D),
self._widget)
        # self.shortcut_shift_d.setContext(Qt.ApplicationShortcut)
        #
self.shortcut_shift_d.activated.connect(self._on_strong_decrease_z_angular_pressed)

        # self.shortcut_space = QShortcut(QKeySequence(Qt.Key_Space), self._widget)
        # self.shortcut_space.setContext(Qt.ApplicationShortcut)
        # self.shortcut_space.activated.connect(self._on_stop_pressed)
        # self.shortcut_space = QShortcut(QKeySequence(Qt.SHIFT + Qt.Key_Space),
self._widget)
        # self.shortcut_space.setContext(Qt.ApplicationShortcut)
        # self.shortcut_space.activated.connect(self._on_stop_pressed)

        #
self._widget.stop_push_button.setToolTip(self._widget.stop_push_button.toolTip() + ' ' +
self.tr('([Shift +] Space)'))
```

```
        #
self._widget.increase_x_linear_push_button.setToolTip(self._widget.increase_x_linear_
push_button.toolTip() + ' ' + self.tr('([Shift +] W)'))
        #
self._widget.reset_x_linear_push_button.setToolTip(self._widget.reset_x_linear_push_b
utton.toolTip() + ' ' + self.tr('([Shift +] X)'))
        #
self._widget.decrease_x_linear_push_button.setToolTip(self._widget.decrease_x_linear
_push_button.toolTip() + ' ' + self.tr('([Shift +] S)'))
        #
self._widget.increase_z_angular_push_button.setToolTip(self._widget.increase_z_angul
ar_push_button.toolTip() + ' ' + self.tr('([Shift +] A)'))
        #
self._widget.reset_z_angular_push_button.setToolTip(self._widget.reset_z_angular_pus
h_button.toolTip() + ' ' + self.tr('([Shift +] Z)'))
        #
self._widget.decrease_z_angular_push_button.setToolTip(self._widget.decrease_z_ang
ular_push_button.toolTip() + ' ' + self.tr('([Shift +] D)'))

        # # timer to consecutively send twist messages
        # self._update_parameter_timer = QTimer(self)
        # self._update_parameter_timer.timeout.connect(self._on_parameter_changed)
        # self._update_parameter_timer.start(100)
        # self.zero_cmd_sent = False

    # @Slot(str)
    # def _on_topic_changed(self, topic):
    #     topic = str(topic)
    #     self._unregister_publisher()
    #     try:
    #         self._publisher = rospy.Publisher(topic, Twist, queue_size=10)
    #     except TypeError:
    #         self._publisher = rospy.Publisher(topic, Twist)

    def _on_set_stepper_changed(self, degrees):
        try:
            self.text_field_angle = int(degrees)
        except Exception, e:
            self.text_field_angle = 0
```

```
def _on_send_stepper_pressed(self):
    self.PC_to_Arduino.stepper_angle = self.text_field_angle
    self._publisher.publish(self.PC_to_Arduino)
    self.PC_to_Arduino.stepper_angle = 0
    self._publisher.publish(self.PC_to_Arduino)

def _on_set_motor_velocity(self,velocity):
    try:
        self.text_field_motor_velocity = int(velocity)
    except Exception, e:
        self.text_field_motor_velocity = 0

def _on_send_motor_velocity(self):
    self.PC_to_Arduino.motor_velocity = self.text_field_motor_velocity
    self._publisher.publish(self.PC_to_Arduino)

def _on_set_motor_angle(self,angle):
    try:
        self.text_field_motor_angle = int(angle)
    except Exception, e:
        self.text_field_motor_angle = 0

def _on_send_motor_angle(self):
    self.PC_to_Arduino.motor_angle = self.text_field_motor_angle
    self._publisher.publish(self.PC_to_Arduino)

def _on_motor_position_checkbox_clicked(self):
    if self._widget.motor_position_checkbox.isChecked():
        self.PC_to_Arduino.motor_position_checked = 1
    else:
        self.PC_to_Arduino.motor_position_checked = 0
    self._publisher.publish(self.PC_to_Arduino)

def _on_motor_speed_checkbox_clicked(self):
    if self._widget.motor_speed_checkbox.isChecked():
        self.PC_to_Arduino.motor_speed_checked = 1
    else:
```

```
            self.PC_to_Arduino.motor_speed_checked = 0
        self._publisher.publish(self.PC_to_Arduino)

    def _on_servo_checkbox_clicked(self):
        if self._widget.servo_checkbox.isChecked():
            self.PC_to_Arduino.servo_checked = 1
        else:
            self.PC_to_Arduino.servo_checked = 0
        self._publisher.publish(self.PC_to_Arduino)

    def _on_thermistor_checkbox_clicked(self):
        if self._widget.thermistor_checkbox.isChecked():
            self.PC_to_Arduino.thermistor_checked = 1
        else:
            self.PC_to_Arduino.thermistor_checked = 0
        self._publisher.publish(self.PC_to_Arduino)

    def _on_light_gate_checkbox_clicked(self):
        if self._widget.light_gate_checkbox.isChecked():
            self.PC_to_Arduino.light_gate_checked = 1
        else:
            self.PC_to_Arduino.light_gate_checked = 0
        self._publisher.publish(self.PC_to_Arduino)

    def _on_ir_checkbox_clicked(self):
        if self._widget.ir_checkbox.isChecked():
            self.PC_to_Arduino.ir_checked = 1
        else:
            self.PC_to_Arduino.ir_checked = 0
        self._publisher.publish(self.PC_to_Arduino)

    def _on_ultra_checkbox_clicked(self):
        if self._widget.ultra_checkbox.isChecked():
            self.PC_to_Arduino.ultra_checked = 1
        else:
            self.PC_to_Arduino.ultra_checked = 0
        self._publisher.publish(self.PC_to_Arduino)
```

```
    # def _on_stop_pressed(self):
    #     self._widget.x_linear_slider.setValue(0)
    #     self._widget.z_angular_slider.setValue(0)

    # def _on_x_linear_slider_changed(self):
    #     self._widget.current_x_linear_label.setText('%0.2f m/s' %
(self._widget.x_linear_slider.value() / MotorlabGUI.slider_factor))
    #     self._on_parameter_changed()

    # def _on_z_angular_slider_changed(self):
    #     self._widget.current_z_angular_label.setText('%0.2f rad/s' %
(self._widget.z_angular_slider.value() / MotorlabGUI.slider_factor))
    #     self._on_parameter_changed()

    # def _on_increase_x_linear_pressed(self):
    #     self._widget.x_linear_slider.setValue(self._widget.x_linear_slider.value() +
self._widget.x_linear_slider.singleStep())

    # def _on_reset_x_linear_pressed(self):
    #     self._widget.x_linear_slider.setValue(0)

    # def _on_decrease_x_linear_pressed(self):
    #     self._widget.x_linear_slider.setValue(self._widget.x_linear_slider.value() -
self._widget.x_linear_slider.singleStep())

    # def _on_increase_z_angular_pressed(self):
    #     self._widget.z_angular_slider.setValue(self._widget.z_angular_slider.value() +
self._widget.z_angular_slider.singleStep())

    # def _on_reset_z_angular_pressed(self):
    #     self._widget.z_angular_slider.setValue(0)

    # def _on_decrease_z_angular_pressed(self):
    #     self._widget.z_angular_slider.setValue(self._widget.z_angular_slider.value() -
self._widget.z_angular_slider.singleStep())

    # def _on_max_x_linear_changed(self, value):
    #     self._widget.x_linear_slider.setMaximum(value * MotorlabGUI.slider_factor)
```

```
    # def _on_min_x_linear_changed(self, value):
    #     self._widget.x_linear_slider.setMinimum(value * MotorlabGUI.slider_factor)

    # def _on_max_z_angular_changed(self, value):
    #     self._widget.z_angular_slider.setMaximum(value * MotorlabGUI.slider_factor)

    # def _on_min_z_angular_changed(self, value):
    #     self._widget.z_angular_slider.setMinimum(value * MotorlabGUI.slider_factor)

    # def _on_strong_increase_x_linear_pressed(self):
    #     self._widget.x_linear_slider.setValue(self._widget.x_linear_slider.value() +
self._widget.x_linear_slider.pageStep())

    # def _on_strong_decrease_x_linear_pressed(self):
    #     self._widget.x_linear_slider.setValue(self._widget.x_linear_slider.value() -
self._widget.x_linear_slider.pageStep())

    # def _on_strong_increase_z_angular_pressed(self):
    #     self._widget.z_angular_slider.setValue(self._widget.z_angular_slider.value() +
self._widget.z_angular_slider.pageStep())

    # def _on_strong_decrease_z_angular_pressed(self):
    #     self._widget.z_angular_slider.setValue(self._widget.z_angular_slider.value() -
self._widget.z_angular_slider.pageStep())

    # def _on_parameter_changed(self):
    #     self._send_twist(self._widget.x_linear_slider.value() / MotorlabGUI.slider_factor,
self._widget.z_angular_slider.value() / MotorlabGUI.slider_factor)

    # def _send_twist(self, x_linear, z_angular):
    #     if self._publisher is None:
    #         return
    #     twist = Twist()
    #     twist.linear.x = x_linear
    #     twist.linear.y = 0
    #     twist.linear.z = 0
    #     twist.angular.x = 0
    #     twist.angular.y = 0
    #     twist.angular.z = z_angular
```

```
    #     # Only send the zero command once so other devices can take control
    #     if x_linear == 0 and z_angular == 0:
    #         if not self.zero_cmd_sent:
    #             self.zero_cmd_sent = True
    #             self._publisher.publish(twist)
    #     else:
    #         self.zero_cmd_sent = False
    #         self._publisher.publish(twist)

    # def _unregister_publisher(self):
    #     if self._publisher is not None:
    #         self._publisher.unregister()
    #         self._publisher = None

    # def shutdown_plugin(self):
    #     self._update_parameter_timer.stop()
    #     self._unregister_publisher()

    # def save_settings(self, plugin_settings, instance_settings):
    #     instance_settings.set_value('topic' , self._widget.topic_line_edit.text())
    #     instance_settings.set_value('vx_max',
self._widget.max_x_linear_double_spin_box.value())
    #     instance_settings.set_value('vx_min',
self._widget.min_x_linear_double_spin_box.value())
    #     instance_settings.set_value('vw_max',
self._widget.max_z_angular_double_spin_box.value())
    #     instance_settings.set_value('vw_min',
self._widget.min_z_angular_double_spin_box.value())

    # def restore_settings(self, plugin_settings, instance_settings):

    #     value = instance_settings.value('topic', "/cmd_vel")
    #     value = rospy.get_param("~default_topic", value)
    #     self._widget.topic_line_edit.setText(value)

    #     value = self._widget.max_x_linear_double_spin_box.value()
    #     value = instance_settings.value( 'vx_max', value)
    #     value = rospy.get_param("~default_vx_max", value)
```

```
#     self._widget.max_x_linear_double_spin_box.setValue(float(value))

#     value = self._widget.min_x_linear_double_spin_box.value()
#     value = instance_settings.value( 'vx_min', value)
#     value = rospy.get_param("~default_vx_min", value)
#     self._widget.min_x_linear_double_spin_box.setValue(float(value))

#     value = self._widget.max_z_angular_double_spin_box.value()
#     value = instance_settings.value( 'vw_max', value)
#     value = rospy.get_param("~default_vw_max", value)
#     self._widget.max_z_angular_double_spin_box.setValue(float(value))

#     value = self._widget.min_z_angular_double_spin_box.value()
#     value = instance_settings.value( 'vw_min', value)
#     value = rospy.get_param("~default_vw_min", value)
#     self._widget.min_z_angular_double_spin_box.setValue(float(value))
```

Main.ino

```
#include <ros.h>
#include <std_msgs/Int16.h>
#include <motorlab_msgs/MotorLab_Arduino.h>
#include <motorlab_msgs/MotorLab_PC.h>
#include "Ultrasonic.h"
#include "IRSensor.h"
#include "LightGate.h"
#include "Servo.h"
#include "Thermistor.h"
#include "StepperMotor.h"

#include "motor_position_controller.h"
#include "motor_velocity_controller.h"
```

```
const int button0_pin = 2;
int but0_old_state = LOW;
int but0_new_state = LOW;
int current_state = 0;

//State Defines
#define NO_SENSE 0
#define THERM_SENSE 1
#define LGATE_SENSE 2
#define IRRNG_SENSE 4
#define ULTRA_SENSE 8

#define NO_ACT 0
#define M_POS_ACT 1
#define M_VEL_ACT 2
#define SERVO_ACT 4


// Actuators
Servo myServo;
StepperMotor myStepper(10, 11);
/******* Begin Motor Stuff **********/
// Pins
const byte motorEncoderAPort = 19;
const byte motorEncoderBPort = 18;
const byte motorL1Port = 6;
const byte motorL2Port = 7;
const byte motorSpeedPort = 12;

MotorController motorController(motorL1Port, motorL2Port, motorSpeedPort);
MotorPositionController* motorPositionController;
MotorVelocityController* motorVelocityController;
Encoder encoder(motorEncoderAPort, motorEncoderBPort);
/******* End Motor Stuff **********/


// Sensors
Ultrasonic ultraSensor(4);
IRSensor irSensor(A1);
```

```
Thermistor tempSensor(A2);
LightGate lightGateSensor(5);

// ROS things
ros::NodeHandle arduinoNode;
motorlab_msgs::MotorLab_Arduino ArduinoMsg;
motorlab_msgs::MotorLab_PC PCMsg;

unsigned long time0 = millis();

void but0_state()
{
  if (but0_new_state == HIGH && but0_old_state == LOW)
  {
    if (millis() - time0 > 100)
    {
      time0 = millis();
      but0_old_state = but0_new_state;
    }
  }
  if (but0_new_state == LOW && but0_old_state == HIGH)
  {
    if (millis() - time0 > 100)
    {
      time0 = millis();
      but0_old_state = but0_new_state;
      current_state++;
      current_state %= 2;
      ArduinoMsg.button_state=current_state*10;
    }
  }
}

void button0_isr_change()
{
  if(but0_new_state==HIGH)
  {
    but0_new_state=LOW;
  }
```

```
  else
  {
    but0_new_state=HIGH;
  }
  but0_state();
}

//Gobal var for Servo Angle
int servo_angle = 0;

//State Variables
int sensor_state = 0;
int actuator_state = 0;

//Vars for mapping
int sensor_max = 0;
int sensor_min = 0;
int sensor_reading = 0;
int actuator_max = 0;
int actuator_min = 0;
int actuator_effort = 0;

void PC_callback(const motorlab_msgs::MotorLab_PC& pc_msg)
{
  if (pc_msg.stepper_angle != 0)
  {
    PCMsg.stepper_angle = pc_msg.stepper_angle;
  }
  PCMsg.motor_position_checked = pc_msg.motor_position_checked;
  PCMsg.motor_speed_checked = pc_msg.motor_speed_checked;
  PCMsg.servo_checked = pc_msg.servo_checked;
  PCMsg.thermistor_checked = pc_msg.thermistor_checked;
  PCMsg.light_gate_checked = pc_msg.light_gate_checked;
  PCMsg.ir_checked = pc_msg.ir_checked;
  PCMsg.ultra_checked = pc_msg.ultra_checked;
  PCMsg.motor_velocity = pc_msg.motor_velocity;
  PCMsg.motor_angle = pc_msg.motor_angle;
}
```

```
ros::Publisher output_msg("ArduinoMsg",&ArduinoMsg);
ros::Subscriber <motorlab_msgs::MotorLab_PC> input_msg("PCMsg",&PC_callback);

/************** Arduino Specific ********************/

void setup() {
  // ROS Initialization
  arduinoNode.initNode();
  arduinoNode.advertise(output_msg);

  arduinoNode.subscribe(input_msg);

  // Initialize Actuators
  myServo.attach(9);
  pinMode(button0_pin, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(button0_pin),button0_isr_change,CHANGE);

    //Motor Stuff
  motorPositionController = new MotorPositionController(&encoder, motorL1Port,
motorL2Port, motorSpeedPort, &motorController);
  motorVelocityController = new MotorVelocityController(&encoder, motorL1Port,
motorL2Port, motorSpeedPort, &motorController);
}

void loop() {
  // package the output message
  ArduinoMsg.dc_motor_position = (*motorPositionController).getAngle();
  ArduinoMsg.dc_motor_speed = (*motorVelocityController).getVeloctiyRPM();
  ArduinoMsg.servo_position = servo_angle;
  ArduinoMsg.stepper_motor_position = 0;
  ArduinoMsg.temperature = tempSensor.gettemperature();
  ArduinoMsg.light_gate_state=lightGateSensor.getState();
  ArduinoMsg.ultrasonic_distance = ultraSensor.pulse_width_measurement();
  ArduinoMsg.ir_distance = irSensor.distanceReading();
  output_msg.publish(&ArduinoMsg);

  //Stepper Handling
  if((PCMsg.stepper_angle != 0) && (current_state != 0)){
    myStepper.moveDegrees(PCMsg.stepper_angle);
```

```
  PCMsg.stepper_angle = 0;
}

//Sensor State
sensor_state = THERM_SENSE*PCMsg.thermistor_checked +
            LGATE_SENSE*PCMsg.light_gate_checked +
            IRRNG_SENSE*PCMsg.ir_checked +
            ULTRA_SENSE*PCMsg.ultra_checked;
switch(sensor_state){
  case THERM_SENSE:
    sensor_min = 400;
    sensor_max = 600;
    sensor_reading = ArduinoMsg.temperature;
    sensor_reading = constrain(sensor_reading, sensor_min, sensor_max);
    break;
  case LGATE_SENSE:
    sensor_min = 0;
    sensor_max = 1;
    sensor_reading = ArduinoMsg.light_gate_state;
    sensor_reading = constrain(sensor_reading, sensor_min, sensor_max);
    break;
  case IRRNG_SENSE:
    sensor_min = 50;
    sensor_max = 600;
    sensor_reading = ArduinoMsg.ir_distance;
    sensor_reading = constrain(sensor_reading, sensor_min, sensor_max);
    break;
  case ULTRA_SENSE:
    sensor_min = 850;
    sensor_max = 5000;
    sensor_reading = ArduinoMsg.ultrasonic_distance;
    sensor_reading = constrain(sensor_reading, sensor_min, sensor_max);
    break;
  default:
    sensor_min = -1;
    sensor_max = -1;
    break;
}
```

```
  //Motor velocit handling when no sensor checked
  if((current_state != 0)&&PCMsg.motor_speed_checked&&(sensor_max == -1) &&
(sensor_min == -1)){
    //motor velocity control code goes here
    (*motorVelocityController).setVelocity(PCMsg.motor_velocity);
  }

  if((current_state != 0)&&PCMsg.motor_position_checked&&(sensor_max == -1) &&
(sensor_min == -1)){
    //motor angle control code goes here
    (*motorPositionController).setAngle(PCMsg.motor_angle);
  }

  //Actuator State
  if ((sensor_max != -1) && (sensor_min != -1) && (current_state != 0))//run only if sensor
selection is valid
  {
    actuator_state = M_POS_ACT*PCMsg.motor_position_checked +
M_VEL_ACT*PCMsg.motor_speed_checked + SERVO_ACT*PCMsg.servo_checked;
    switch(actuator_state){
      case M_POS_ACT:
        actuator_min = 60;
        actuator_max = 300;
        actuator_effort = map(sensor_reading, sensor_min, sensor_max, actuator_min,
actuator_max);
        (*motorPositionController).setAngle(actuator_effort);
        break;
      case M_VEL_ACT:
        actuator_min = -70;
        actuator_max = 70;
        actuator_effort = map(sensor_reading, sensor_min, sensor_max, actuator_min,
actuator_max);
        (*motorVelocityController).setVelocity(actuator_effort);
        break;
      case SERVO_ACT:
        actuator_min = 0;
        actuator_max = 180;
```

```
        actuator_effort = map(sensor_reading, sensor_min, sensor_max, actuator_min,
actuator_max);
        servo_angle = actuator_effort;
        myServo.write(actuator_effort);
        break;
       default:
        myServo.write(servo_angle);
        (*motorVelocityController).setVelocity(0);
        break;
    }
  }
  else{
    myServo.write(servo_angle);
    if((current_state==0))
    {
      (*motorVelocityController).setVelocity(0);
    }
  }



  arduinoNode.spinOnce();
  delay(100);
}
```

motorlab_translator.py

```
#!/usr/bin/env python

import rospy
import math
from std_msgs.msg import Bool
from motorlab_msgs.msg import MotorLab_Arduino
from motorlab_msgs.msg import MotorLab_Arduino_Translation

class Arduino_Translator(object):
        """docstring for Arduino_Translator"""
        def __init__(self):
                self.Arduino_result = MotorLab_Arduino()
                self.Arduino_translation = MotorLab_Arduino_Translation()
```

```
		self.arduino_sub = rospy.Subscriber("ArduinoMsg", MotorLab_Arduino,
self.ArduinoCb)
		self.arduino_pub = rospy.Publisher("ArduinoTranslated",
MotorLab_Arduino_Translation, queue_size = 1)
		self.light_gate_pub = rospy.Publisher("LightGateState", Bool, queue_size
= 1)
		self.button_pub = rospy.Publisher("ActuatorsOn", Bool, queue_size = 1)

	def ArduinoCb(self,data):
		self.Arduino_result = data

	def publishArduino(self):
		self.Arduino_translation.dc_motor_position =
self.Arduino_result.dc_motor_position
		self.Arduino_translation.dc_motor_speed =
self.Arduino_result.dc_motor_speed
		self.Arduino_translation.servo_position =
self.Arduino_result.servo_position
		self.Arduino_translation.stepper_motor_position =
self.Arduino_result.stepper_motor_position
		self.Arduino_translation.temperature =
self.translate_temp(self.Arduino_result.temperature)
		self.Arduino_translation.light_gate_state =
self.Arduino_result.light_gate_state
		self.Arduino_translation.ultrasonic_distance =
self.translate_ultra(self.Arduino_result.ultrasonic_distance)
		self.Arduino_translation.ir_distance =
self.translate_IR(self.Arduino_result.ir_distance)
		self.Arduino_translation.button_state = self.Arduino_result.button_state
		self.light_gate_pub.publish(bool(self.Arduino_result.light_gate_state))
		self.button_pub.publish(bool(self.Arduino_result.button_state))
		self.arduino_pub.publish(self.Arduino_translation)

	def translate_temp(self,rawADC):
		Temp = 0.0
		if rawADC > 0:
			# Temp = math.log(10000.0*((1024.0/rawADC-1)));
			# Temp = math.log(10000.0/(1024.0/rawADC-1)) # for pull-up
configuration
```

```
                        # Temp = 1.0 / (0.001129148 + (0.000234125 + (0.0000000876741
* Temp * Temp ))* Temp );

                        # Temp = Temp - 273.15;           # Convert Kelvin to Celcius
                        # Temp = (Temp * 9.0)/ 5.0 + 32.0; # Convert Celcius to Fahrenheit
                        Temp = 0.0968*rawADC-22.943
                        Temp = (Temp * 9.0)/ 5.0 + 32.0; # Convert Celcius to Fahrenheit
                else:
                        Temp = -300
                return Temp;

        def translate_IR(self,rawADC):
                dist = -0.072*rawADC+41.531
                if dist > 40.0:
                        return 40.0
                return dist

        def translate_ultra(self, pulseWidth):
                dist = (pulseWidth*0.013)-4.4 #inches
                if dist > 40.0:
                        return 40.0
                return dist

if __name__ == '__main__':

  # Init Node
  rospy.init_node('Arduino_Translator')

  ATranslator = Arduino_Translator()

  rate = rospy.Rate(120) #hz
  while not rospy.is_shutdown():
    ATranslator.publishArduino()
    rate.sleep()
```