Individual Lab Report #1

Sensors and Motors Lab

February 13, 2025

Jet Situ

Team B

Teammates: Gweneth Ge, Lance Liu, Yi Wu, Joshua Pen

Table of Contents

Contents

1 Individual Progress

1.1 Sensors and Motor Lab

1.1.1 My Contribution

My contribution to the project was in two distinct, but related sections - the integration of the numerous segments of code, and the UI. In the prior role, I worked with Wuyi in order to separate out the distinct sensor/motor combinations into a presentable format.

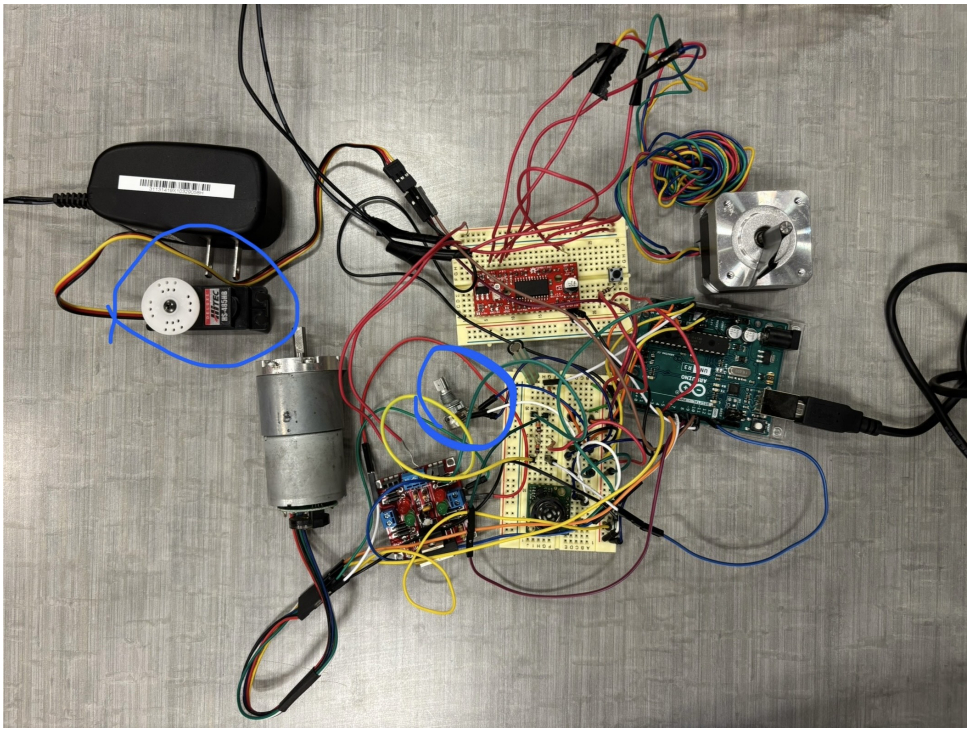

Figure 1: The Full Circuit Layout

Above is the figure of the completed circuit. To divide up the motors, which can have issues with cross-interference during their own phases, and to make distinct the chain of actions, Wuyi and I worked together to divide the system into 6 distinct modes. These distinct modes allowed us to demonstrate individual sensor/motor combinations that showcased the range of motion and range of sensing, and our ability to integrate them. In this role, I took on more of an architectural/design role, working on the mode switches and the back and forth commands needed between the Uno and the GUI. I also worked in a debugging role, as integration created significant problems in the operation of the start and stop of some of the motors, which had to be addressed through automatic stops and switches upon certain mode changes.

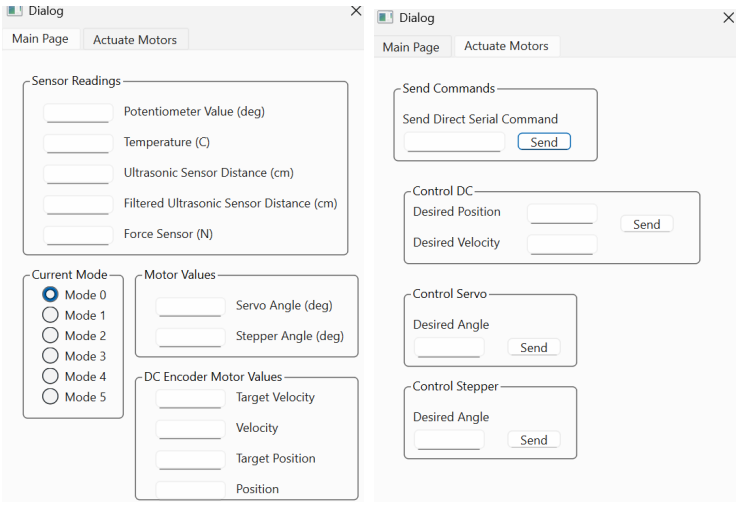

Figure 2: The GUI

My next contribution was toward the GUI (done in Qt), and the interfacing between the Arduino and the GUI. Primary work on this was done in conjunction with the integration phase, where for each mode, I added serial print commands to be sent back to the main computer, containing the values relevant to that mode, including, but not limited to the potentiometer reading, the temperature reading, the ultrasonic and filtered ultrasonic readings, the force sensor readings, and for the motor section, the positions and active PID control for each motor. These had to abide by a specific format in order to be read by the main computer, but the format was standardized, consisting of a single line separating parameter and value. The incoming values were displayed on the main page of the GUI, for user reference.

The next part of the GUI was the control segment, which I wrote on the Arduino, binding these functions to Mode 5. The format is the same as incoming messages, so outbound messages are read in by the Arduino, which then does various functions. For debugging purposes, I added in the function to send a direct serial command, but for user experience, I also added in the functions to directly control the DC motor, the servo motor, and the stepper motor. For each one, both positive and negative commands work to actuate each motor to its full possible range of motion, in both the clockwise and counterclockwise direction.

During demonstration, the GUI is designed to show a direct feedback loop - values are automatically propagated and reflected back in the main page, as a verification step for the currently running mode. This made the overall GUI user-friendly and intuitive to use, even to first-time users.

1.2 The GUI Code

1.2.1 dialog.h

```
#define DIALOG_H

#include <QDialog>
#include <QSerialPort>
#include <QSerialPortInfo>
#include <QDebug>
```

```
#include <QtWidgets>

QT_BEGIN_NAMESPACE
namespace Ui {
class Dialog;
}
QT_END_NAMESPACE

class Dialog : public QDialog
{
    Q_OBJECT

public:
    Dialog(QWidget *parent = nullptr);
    ~Dialog();

private slots:
    void readSerialData();
    void sendSerialData();
    void sendSerialDCData();
    void sendSerialServoData();
    void sendSerialStepperData();

private:
    Ui::Dialog *ui;
    //required setup
    QSerialPort *arduino;
    QString arduino_port_name;
    static const quint16 arduino_uno_vendorID = 9025;
    static const quint16 arduino_uno_productID = 67;
    bool arduino_is_available = false;

    //for sensors
    qint16 potReading;
    qint16 tempReading;
    qfloat16 ultrasonicReading;
    qfloat16 filteredUltrasonicReading;
    qfloat16 forceReading;

    //for servo
    qfloat16 servoReading;

    //for the mode
    qint8 mode;
};
#endif // DIALOG_H
```

1.2.2 dialog.cpp

```
#include "dialog.h"
#include "ui_dialog.h"

Dialog::Dialog(QWidget *parent)
    : QDialog(parent)
    , ui(new Ui::Dialog)
{
    ui->setupUi(this);

    //setup here
    arduino = new QSerialPort;

    qDebug() << "Number of Ports: " << QSerialPortInfo::availablePorts().length();
    foreach (const QSerialPortInfo &serialPortInfo, QSerialPortInfo::availablePorts()
        if(serialPortInfo.hasVendorIdentifier() && serialPortInfo.hasProductIdentifie
            qDebug() << "Vendor ID: " << serialPortInfo.vendorIdentifier();
            qDebug() << "Product ID: " << serialPortInfo.productIdentifier();

            if(serialPortInfo.vendorIdentifier() == arduino_uno_vendorID &&
                serialPortInfo.productIdentifier() == arduino_uno_productID) {
                arduino_port_name = serialPortInfo.portName();
                arduino_is_available = true;
            }
        }
    }

    if(arduino_is_available) {
        arduino->setPortName(arduino_port_name);
        arduino->setBaudRate(QSerialPort::Baud9600);
        arduino->setDataBits(QSerialPort::Data8);
        arduino->setParity(QSerialPort::NoParity);
        arduino->setStopBits(QSerialPort::OneStop);
        arduino->setFlowControl(QSerialPort::NoFlowControl);
        arduino->open(QSerialPort::ReadWrite);

        connect(arduino, &QSerialPort::readyRead, this, &Dialog::readSerialData);
    } else {
        QMessageBox::warning(this, "Port error", "Arduino not found");
    }

    connect(ui->pushSend,&QPushButton::clicked, this, &Dialog::sendSerialData);
    connect(ui->pushSendDC,&QPushButton::clicked, this, &Dialog::sendSerialDCData);
    connect(ui->pushSendServo,&QPushButton::clicked, this, &Dialog::sendSerialServoDa
    connect(ui->pushSendStepper,&QPushButton::clicked, this, &Dialog::sendSerialStepp
    ui->radioMode0->setChecked(true);
}
```

```
Dialog::~Dialog()
{
    if(arduino->isOpen()) {
        qDebug() << "closing port";
        arduino->close();
    }
    delete ui;
}


QString buffer;
void Dialog::readSerialData() {
    buffer += QString::fromUtf8(arduino->readAll());

    int endIndex;
    while ((endIndex = buffer.indexOf('\n')) != -1) {
        QString message = buffer.left(endIndex).trimmed();
        buffer.remove(0, endIndex + 1);

        qDebug() << "Complete message received:" << message;
        //string split here
        QStringList parts = message.split(':');
        if (parts.size() != 2) {
            qDebug() << "Invalid message format:" << message;
            return;  // Skip if the format is incorrect
        }

        QString label = parts[0].trimmed();
        QString value = parts[1].trimmed();

        if (label == "POT") {
            ui->linePotVal->setText(value);
        }
        if (label == "TEMP") {
            ui->lineTemp->setText(value);
        }
        if (label == "SERVO") {
            ui->lineServo->setText(value);
        }
        if (label == "ULTRA") {
            ui->lineUltra->setText(value);
        }
        if (label == "FILTERULTRA") {
            ui->lineFilterUltra->setText(value);
        }
        if (label == "FORCE") {
            ui->lineForce->setText(value);
```

```
        }
        if (label == "STATE") {
            ui->radioMode0->setChecked((value.toInt() % 6 == 0));
            ui->radioMode1->setChecked((value.toInt() % 6 == 1));
            ui->radioMode2->setChecked((value.toInt() % 6 == 2));
            ui->radioMode3->setChecked((value.toInt() % 6 == 3));
            ui->radioMode4->setChecked((value.toInt() % 6 == 4));
            ui->radioMode5->setChecked((value.toInt() % 6 == 5));
        }
        if(label == "STEPANGLE") {
            ui->lineStepper->setText(value);
        }
        if(label == "TARGETVEL") {
            ui->lineTargetVelocity->setText(value);
        }
        if(label == "VEL") {
            ui->lineVelocity->setText(value);
        }
        if(label == "POS") {
            ui->linePosition->setText(value);
        }
        if(label == "TARGETPOS") {
            ui->lineTargetPosition->setText(value);
        }
    }
}

void Dialog::sendSerialData() {
    QString message = ui->lineSend->text().trimmed();
    if (arduino->isOpen() && arduino->isWritable()) {
        arduino->write(message.toUtf8());
        arduino->write("\n");
        qDebug() << "Sent";
        ui->lineSend->clear();
    }
}

void Dialog::sendSerialDCData() {
    QString velocity = ui->lineDesVel->text().trimmed();
    QString position = ui->lineDesPos->text().trimmed();
    if (arduino->isOpen() && arduino->isWritable()) {
        if(velocity != "") {
            arduino->write("DCVEL:");
            arduino->write(velocity.toUtf8());
            arduino->write("\n");
            ui->lineDesVel->clear();
        } else {
            arduino->write("DCPOS:");
```

```
            arduino->write(position.toUtf8());
            arduino->write("\n");
            ui->lineDesPos->clear();
        }
    }
}

void Dialog::sendSerialServoData() {
    QString position = ui->lineDesServo->text().trimmed();
    if (arduino->isOpen() && arduino->isWritable()) {
        arduino->write("SERVO:");
        arduino->write(position.toUtf8());
        arduino->write("\n");
        ui->lineDesServo->clear();
    }
}

void Dialog::sendSerialStepperData() {
    QString position = ui->lineDesStepper->text().trimmed();
    if (arduino->isOpen() && arduino->isWritable()) {
        arduino->write("STEPPER:");
        arduino->write(position.toUtf8());
        arduino->write("\n");
        ui->lineDesStepper->clear();
    }
}
```

1.3 MRSD Project

My role in the MRSD project was to serve as the primary liason between the AirLab's goal for the DARPA Triage Challenge, and the MRSD team, as well as the primary designer of the electrical and overall software and communication architecture for the drone. To that end, my primary goal was coordinating and constructing the drone systems to make it fit to participate in the DARPA Triage Challenge Workshop 2, which is the workshop being held in March in Perry, Georgia, at the onsite DARPA location. However, this means that we had to abide by an accelerated DARPA timeline, which made our work accelerated in comparision to the standard MRSD schedule.

Specifically, we had a qualification safety deadline by January 15th to demonstrate the operation of our drone's autonomous flight capability, and e-stop capability. Only in December were we able to begin active work on the drone, which consisted of several build phases. We first began by deconstructing the DJI Matricie 100 drone, which serves as the mechanical base of the drone. However, to comply with DARPA NDAA requirements, we removed all antennas, transmission devices, and computing capability from the Matricie 100. In the next phase, we took an Aurelia X6 drone, and removed its computing capability, GNSS antenna, transmitter, and step-down converter. Because these parts were NDAA compliant, we were able to then transfer these components, and integrate them onto the DJI drone, which I headed the effort for. As part of our third build phase, we then routed the power from the DJI battery to the new subcomponents, powering the radio, GPS, the Cube Blue autopilot, and the gimbal device. Following that, we ran a payload flight test, verifying that the drone's stabilizing controllers could handle the increased payload in its configuration.

My next role then consisted of organizing the test flight at NREC, and integrating the QGroundControl interface that we used for the initial autonomous test flight. I worked with Lance to achieve this, setting up the QGroundControl interface, flight plan, geofence, and MavLink communication to the drone. Finally, we were able to take a video of the drone in action, sending it to DARPA, and qualifying for the first workshop.

After the January 15th deadline, my work on the electrical side has largely remained the same, this time integrating the Orin with the main system, including the routing to the radio, and the routing of the power distribution system to the Orin, drawing on the existing breakout board present on the DJI. My most recent work has focused on the communication protocols between the ground station, the Orin, and the onboard Cube Blue, which has been a challenging communication task, as it operates both on the ROS network and the MavLink protocol, with the transmission device over a remote radio system.

2 Challenges

2.1 Sensors and Motor Lab

The largest challenge encountered during the lab was the DC motor, which had a finicky encoder system that remained difficult to document and predict its behavior. Nailing down the PID proved challenging, and once that was nailed down, the motor's powering system proved to also have a significant power draw. This means that not only did the motor's system require a careful amount of code to run with the proper designed parameters, the DC motor's on and off state must also be carefully timed and run to not

interfere with the stepper motor, which ran off the same 12V source.

This meant that integration proved tricky when operating in tandem with both the stepper and the DC motor. To resolve this, we put both motors onto separate modes. We further delineated these motors by having the action of one automatically shut down the action of the other. Furthermore, the PID positional controller of the motor had to be integrated with a stop command upon reaching an epsilon deviation of the desired state, which safely stopped the motor running, reducing the power draw from the system.

The other challenge we faced during the sensor lab was component failure - we had a stepper motor failure, alongside a microcontroller failure as well, which had to be individually replaced. This cost us time, but due to the backups we made, we were able to quickly retest on newer devices, and overcome this specific challenge.

2.2 MRSD Project

The largest beginning challenge began before the drone was even constructed - the selection of components. Due to NDAA compliance being a dominating factor during the procurement process, we were unable to have access to a wide array of cheaper, and in many cases, more consistent devices for use on our drone. Ruling out the vast majority of manufactured drones forced us to build our own, and this downselection also impacted our choice of radio, firmware, autopilot, and localization hardware, which had a significant risk of lead times and cost overruns.

The next challenge we faced was during flight testing. Because of the unproven nature of our drone, we had to run both a payload integration test, validating that our drone was capable of performing stable in-air flight without direct human intervention at all times, and a fully autonomous flight in an outdoor area in a manner that would not jeopardize the life and safety of the people within the area.

However, these flight tests would prove to be costly - a mishap during the payload integration test lost us a propeller, and a failed flight test during the autonomous flight testing at NREC also lost us some more propellers. Luckily, we had backups, and we ordered more backups as a precaution against further losses. This challenge informed us that the point of failure that ended up being most likely was the propellers - which, as a non-electronic device, we completely overlooked, thinking that we had plenty of spares. To mitigate this risk, we installed prop shields on our drone, both improving its safety to the people on the ground, as well as reducing the risk of a prop break again, which would be significantly more expensive and harder to procure.

3 Team Work

3.1 Sensors and Motor Lab

Name	Sensor	Motor	Contribution
Jet Situ	GUI		GUI implementation with Arduino Integration.
Joshua Pen	Temperature Sensor	DC Stepper	Implemented a user-operated switch with debouncing. Integrated Temperature Sensor with DC Stepper.
Lance Liu	Ultrasonic Range Finder	DC Motor	Integrated Ultrasonic Sensor with Mean Filter and DC Motor. Designed PID controller for DC Motor Postion Control.
Gweneth Ge	Potentiometer	RC Servo	Integrated Potentiometer with RC Servo.
Yi Wu	Force Sensor	DC Motor	Designed PID controller for DC Motor Velocity Control.

Table 1: Team Members and Their Components

3.2 MRSD Project

Name	Contribution
Jet Situ	Assisted in rebuilding the drone for NDAA compliance during the final assembly phase, including mounting components to the provided attachment mounts. Soldered connection wires between the Orin and the Cube Blue, and connected the gimbal's UART control system to the Cube Blue. Validated and connected the gimbal's data connection to onboard networked protocol. Assisted in the development of deployed software on the Orin, and the design of the overall software architecture protocol. Contributed to setting up the ROS2 architecture for inter-system communication in the DTC. Obtained a Part 107 license for purposes of outdoor flight evaluation and testing.
Joshua Pen	Assisted in rebuilding the drone for NDAA compliance, replacing the DJI controller with Cube Blue ArduPilot. Soldered wires to connect various components during the drone rebuild. Designed attachment mounts for the Doodle Labs Radio antennas, Intel Realsense D435, and a separate mount for the NVIDIA Orin NX and Doodle Labs Smart Radio. Developed a mounting solution for the Hadron 640R Gimbal and designed extension legs for the drone. Procured propellers and shields. Contributed to the payload integration and configuration of the Hadron 640R with Cube Blue ArduPilot and NVIDIA Orin NX.
Lance Liu	Contributed to the rebuilding of the NDAA compliant drone. Established communication between the ground control station (GCS) and the drone via a sophisticated radio system. Executed iterative refining and testing on the embedded system. Integrated and validated the interaction pipeline between the GCS and the onboard Orin using MAVROS. Designed a behavior tree to manage decision during the challenge operation. Migrated the AirLab Docker environment to support the ARM architecture on the NVIDIA Orin, and integrated—while continuing to adapt—AirLab's codebase within our platform.
Gweneth Ge	Primarily contributed to the development of the first-version drone for submitting video documentation to DARPA, focusing on sourcing and integration of NDAA-compliant components. Helped the electrical integration, including configuring the Pixhawk ArduPilot and BlackCube on the DJI M100 frame, as well as powering and connecting the gimbal, cameras and radio systems. Additionally, supported overall project management and logistics.
Yi Wu	Collected human detection datasets specifically for drone applications and conducted a literature review of 3D Human Pose Estimation (HPE) algorithms. Collaborated with the AirLab human detection team to test the state-of-the-art HPE algorithm in their x86 and Jetson Orin docker environments, wrapping the algorithm as ROS2 nodes.

Table 2: Team Members and Their Contributions

4 Plans

4.1 MRSD Project

Name	Contribution
Jet Situ	Polygon covering waypoints generator, including sending entire waypoints in one go and flying to designated position at a certain altitude. Gimbal control protocol and sensor nodes development. Take off/landing planner, patients searching logic, task allocation planner, and visualization.
Joshua Pen	In my future role, I will focus on developing gimbal control protocols and sensor nodes, along with additional mechanical modifications. I will assist in sensor nodes development, including detection launch, visualization, and clicking interaction. Furthermore, I will handle project management and logistics.
Lance Liu	ROS2 network refinement. Data transmission of sensors including RGB, Thermal and gimbal. Behavior tree executive and management implementation including auto takeoff, land, safe landing, RTB, mapping, searching, and inspecting. Overall robot system bringing up.
Gweneth Ge	Primarily work on Inter-UAV collision logic and planner launch. Assist in sensor nodes development, detection launch, visualization and clicking interaction. Continue supporting on project management and logistics.
Yi Wu	Integrate the human pose estimation algorithm with the upstream person Re-Identification (ReID) algorithm. Test the algorithm performance on DARPA datasets, and prepare new annotated datasets if necessary for re-training purposes. Develop solutions for pose estimation algorithms using thermal camera data during nighttime conditions.

Table 3: Team Members and Their Plans

5 Sensors Quiz

5.1 Question 1

5.1.1

The sensor's range is -3.6 to 3.6g, where g refers to the force of gravity.

5.1.2

The dynamic range then, is the total range, which is 7.2g.

5.1.3

The capacitor in this case acts as a form of a low-pass filter. It achieves this by averaging out sudden noise shocks in the form of a small microfarad storage, which absorbs small noises that would otherwise produce significant swings within the accelerometer output. High frequency passes will be absorbed by the capacitor, not being picked up in the output voltage.

5.1.4

$V_out = 300a + 1.5$, where a refers to the current acceleration measured by the sensor.

5.1.5

Nonlinearity is expected at 0.0216g. This is because the nonlinearity is at 0.3% of the full scale, so $0.3\%(7.2) = 0.0216$

5.1.6

1600Hz

5.1.7

$(150\mu g/\sqrt{Hz})(5\sqrt{Hz})(10^{-6}) = 0.75mg$

5.1.8

The best way to determine the RMS noise experimentally is to subject the accelerometer to standard gravity. Leave the accelerometer on the table in the x orientation, then the y orientation, then the z orientation, and measure the output voltage for each pin. Because standard gravity is left uncalibrated, we can then measure the output voltage with an oscilloscope, which can then tell us the amplitude of the fluctuation in the voltages. We can then measure that to get our RMS. The assumption here is that the accelerometer is not already calibrated, or set to an offset that negates gravity, and that the testing environment is sterile, so no sources of outside interference can happen.

5.2 Question 2

5.2.1 Average Filter

The first problem in a moving average filter is the balancing act between the averaging filter and the need for specific datapoints - if the moving average filter averages over too long of a timeframe, then critical datapoints that needed to be obtained would be otherwise filtered out.

The second problem is that moving average filters would remain susceptible to high-frequency, high amplitude noisy inputs. Because there isn't a separate filter prior to the

moving average filter, these noises can still make it in, significantly biasing the existing data.

5.2.2 Median Filter

Like the average filter, the first problem with a median filter is that it can fail to capture data if spread across too wide of a timeframe. Data that is important, or needing to be logged can be buried underneath a larger amount of signals, losing the insight that was necessary to be seen.

The second problem with a median filter lies in a non-normal or skewed distribution. In this specific case, while more significant data can be present on one end, the median could filter out these insights, since those datapoints would be equally weighted as datapoints on the other side of the median. For skewed distributions, a lot of the insight on the distribution would be filtered out or lost.

5.2.3 Op-Amp Design

Based on the op-amp design, the equations for the voltage is:

$$\frac{V_1 - V_2}{R_i} = \frac{V_2 - V_{out}}{R_f}$$

5.2.4 -1.5V to 1.0V

In this design, V_2 should be the input, and V_1 should be at -3V. Therefore, at -1.5V, V_{out} = 0, and $R_i = R_f$, so a ratio of 1. At 1.0V, V_out must equal 5V.

5.2.5 -2.5V to 2.5V

This isn't possible to be calibrated. Let V_1 be the input voltage, in this case, R_f/R_i would be -1. Let V_2 be the input voltage, but that would mean that $R_f/R_i = 0$.

5.3 Question 3

5.4

Calculate the error at each timestep, which is the positional delta between desired and actual position (this would require an encoder of some kind to track the position). Then, tune a Kp, Kd, and Ki values, and punch in the positional error and multiply it by Kp. In addition, to the derivative term, add a velocity error, which is equal to the current velocity, since the desired ending velocity is 0. Finally, integrate the position, and multiply that Ki, to achieve the integral term. With tuned values, that equals the input to feed to the motor, and repeat until converged.

5.5

If it's sluggish, that means it's rise time is slow. Increasing Kp will make the gains go much faster, making the system more aggressive overall.

5.6

If there's significant steady-state error, that means that there's an accumulation error. To mitigate errors caused by accumulation, tune Ki to correctly adjust for integral drift.

5.7

If there's too much overshoot, there's two ways of fixing this. The direct way is to increase Kd, which dampens the velocity of the system, which can avoid overshoots. That is the main way. The second alternative way is to reduce Kp, which will have other consequences, such as increasing rise time.