Stabilization of Uni-Directional Mini Segway

Gustav Arvidsson, Nils Hallerfelt, Teodor Westholm, Matti Ågren

Supervisor: Julia Adlercreutz

March 2023

1 Introduction

The MinSeg is a small Segway-process that is built using, and programmed with, an Arduino. The task is to implement a stabilizing controller for this process. The system will use wireless Bluetooth communication to a GUI that resides on a desktop where signals are plotted and setpoints can be changed.

The drivetrain of the MinSeg (model M1V4) is a Lego NXT DC motor equipped with wheels. An Arduino Mega 2560 microcontroller board drives the motor and reads sensor data. The MinSeg will be connected to a PC via a USB. The MinSeg is equipped with two sensor units. The first sensor unit is a rotational encoder in the Lego NXT motor which measures the wheels rotational position around the wheel axis. The second sensor is an IMU (inertial measurement unit). The IMU sensor unit consists of two sensors: an accelerometer and a gyrometer. The accelerometer measures the acceleration along the coordinate axes, while the gyrometer measures the angular velocities around the coordinate axes.

1.1 Programming Languages

In the project, the Arduino programming language and Java is used. Matlab with simulink will also be used, however only during the design and testing of the regulator. Arduino IDE (Integrated Development Environment) is a software application that is used to write, compile, and upload code to Arduino microcontroller boards. It is a cross-platform application that is available for Windows, Mac OS X, and Linux. The Arduino programming language is based on C/C++, but it includes a simplified syntax and a range of built-in functions and libraries that are specific to the Arduino platform. The IDE supports a Bluetooth connection between the board and PC.

Java is used for the on-PC programming. This amounts to setting up a GUI, that allows changes in real-time of set-points and for real-time plotting of control signals. It is thus a real-time communication process with the MinSeg. As a consequence Real-Time Java support is needed.

For the Java program the following packages are imported. The javax.swing library for the GUI, the se.lth.control library for plotting signals and retrieving user inputs as well as the bluecove 2.1.1 library for bluetooth communication with the segway.

1.2 Hardware Specifications

The Arduino Mega 2560 is a microcontroller board for the microcontroller ATmega 2560. It has 54 digital input/output pins (of which 15 can be used as pulse width modulation outputs), 16 analog inputs, 4 UARTs (hardware serial ports), a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP (In-Circuit Serial Programming) header, and a reset button.

1. Operating voltage: 5V
2. Recommended input voltage: 7-12V
3. Limit input voltage: 6-20V
4. Flashmemory: 256KB, of which 8KB is bootlander.

The motor of the MinSeg is a LEGO Mindstorms NXT, which can run at full power when connected to the battery pack. This increases the operating voltage from 5V to 9V. The Lego NXT motor is connected to a MinSegShield M1V4.3 on top of the Arduino Mega. This custom shield will also have the MPU6050 sensors and Bluetooth module. The Bluetooth module used is the Arduino HC-06. This is a simple module with four pins, two for power, one for receiving and one for transmitting data. The HC-06 has a red diode that blinks when it has power, and shine constantly when a connection has been established.

2 Program Structure and GUI

The overall program structure is shown in Figure 1. The control algorithm is implemented on the Arduino board. In this thread, an observer is implemented, the control signal is calculated and applied to the motor and the measurements are communicated to a Java program via Bluetooth. On a separate computer, an OpCom class, which contains two Plotter threads, initializes the GUI. This class also contains a private class ArduinoMonitor, that handles the communication with the Arduino. Besides this, a class Plotter with its own thread, reads the values sent from the Arduino and sends them on to the OpCom class to be plotted. Finally a ReferenceGenerator class, also with its own thread, updates the reference signal and sends it to the Arduino via the OpCom class. The whole process is started by the StartUp class with a main method that will initialize the GUI.

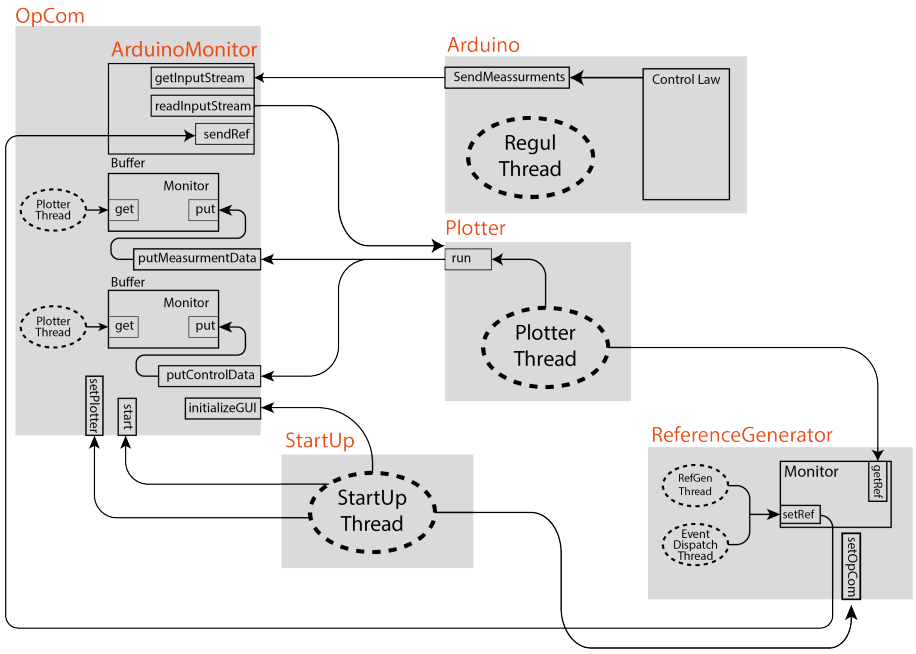

Figure 1: A structure diagram of the program.

A user interface for controlling the MinSeg is implemented using Swing in Java. This is done in the class OpCom that initializes the GUI, plots the control signal, the DC voltage applied to the motor, the MinSeg position which is the output of the process and the position reference. The MinSeg- and reference position are plotted in one graph and the control signal in another. At first, the reference position is 10 radians straight ahead. The amplitude of the reference can be changed by the user from the same window. A separate thread, ReferenceGenerator handles the reference, which changes sign every 10 seconds so that the MinSeg moves back and forth. Because state feedback is used for the controller, sudden reference changes will create a spike in the

control signal, which risks causing the MinSeg to lose balance. Therefore the reference is changed over time with a speed that the MinSeg can handle.

The GUI window is implemented by a JFrame that is a private attribute to the OpCom class. In this frame, two PlotterPanels are used to plot the control signal and the segway position and reference respectively. Below this a separate JPanel will contain the reference toggle from default to manual mode in the form of two JRadioButtons, one JButton as start button as well as the reference amplitude input in the form of a DoubleField. An illustration of this interface is shown in figure 2.

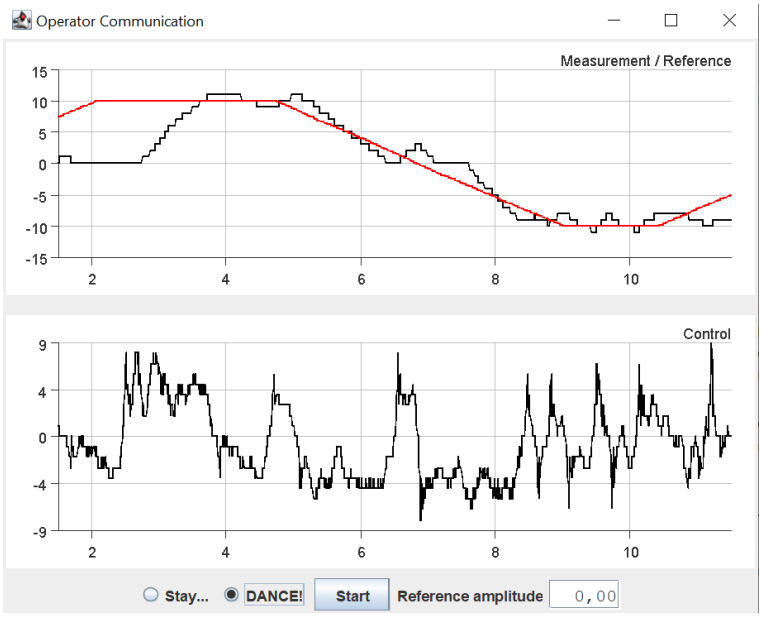

Figure 2: The GUI window implemented using Swing.

2.1 How to

When the MinSeg is turned on, it has a position reference set to zero, so that the MinSeg will balance on its own before the communication with the computer is begun. The program is started by running the StartUp program. This will start the initializing the Bluetooth connection. If it succeeds, the GUI will be initialized and appear on the screen as seen in 2. When the the start button on the main screen is pressed, the Plotter and ReferenceGenerator threads will be started so that the program will start receiving measurement signals from the MinSeg, as well as send a reference back so that the MinSeg starts moving forward. All the while the MinSeg is moving, its position versus the reference signal, as well as the control signal is plotted.

2.2 Bluetooth connection

For the GUI it is a requirement to send and receive information from the MinSeg. The user should be able to change the reference, meaning that this change should be sent to the MinSeg. The user should also see the current positon and the control signal from the MinSeg, meaning that this information should be sent from the MinSeg to the java program. This can be done through a cable connection or, as have been chosen here, through Bluetooth.

The first step is to establish a Bluetooth connection. Because the connection must work on both ends for any data to get transmitted, it was hard to know what the problem was if no data was transmitted. First, assume that everything works on the Java side. The HC-06 bluetooth module has a prepared place on the MinSegShield M1V4.3. Looking through documentation on their website it seemed that the reception and transmission pins were D1 and D0. Using Arduino *SoftwareSerial* to read and write to these pins, there was not any sign that anything was transmitted to the computer. After testing a few other options the normal *Serial* ports worked. The assumption is that the pins were connected to the standard TX and RX pins, meaning that it was sent to *Serial*.

On the other end a java Bluetooth package called bluecove was used. From this library io.InputStream, io.OutputStream, io.Connector and io.StreamConnection was imported. Connector was used to establish a connection to the url of the HC-06. This connector was then used to make a *StreamConnection*. Then data was read from *InputStream* and printed to *OutputStream* and the communication between the two was finished.

This connection could only transmit bytes, meaning that some transformations and rounding was needed to transmit the correct values.

3 Model and control principles

3.1 State space model

The process can be modeled as an inverted pendulum with four relevant state variables; the tilt angle α of the MinSeg away from the upright position and the rotational angle θ of the wheel, as well as their derivatives $\dot{\alpha}$ and $\dot{\theta}$. By setting up and solving the equations for force and torque, a non-linear system of differential equations is obtained:

$$\dot{x} = [\ddot{\alpha}, \dot{\alpha}, \ddot{\theta}, \dot{\theta}]^T = f(\dot{\alpha}, \alpha, \dot{\theta}, \theta, u)$$

which is then linearized to the form

$$\dot{x} = Ax + Bu$$

where u is the control signal, i.e the voltage over the motor. Fortunately, the work of finding the matrices A and B had already been done, and the following were used:

$$A = \begin{pmatrix} -3.1 & 58.4 & 62.7 & 0 \\ 1 & 0 & 0 & 0 \\ 40.1 & -318 & -766 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 148 \\ 0 \\ 1808 \\ 0 \end{pmatrix}. \tag{1}$$

Furthermore, the controller would be implemented in discrete time, so the system had to be sampled. A sampling time of 10ms was used, and yielded the following state space matrices:

$$\Phi = \begin{pmatrix} 0.9995 & 0.3569 & 0.0817 & 0 \\ 0.0100 & 1.0019 & 0.0007 & 0 \\ 0.0487 & -0.3991 & 0.0045 & 0 \\ 0.0004 & -0.0035 & 0.0013 & 1.0000 \end{pmatrix}, \quad \Gamma = \begin{pmatrix} -0.1930 \\ -0.0017 \\ 2.3497 \\ 0.0205 \end{pmatrix}. \tag{2}$$

3.2 Measurements

To implement the control laws measurements of the states $\dot{\alpha}, \alpha, \dot{\theta}, \theta$ are needed. The direct measurements that can be obtained are the following:

- The angular rate $\dot{\alpha}$ can be obtained from the gyrometer on the IMU. These measurements are noisy.
- The tilt angle α can be constructed from the accelerometer with some geometry. This measurement is noisy, especially when the process is moving fast.
- The wheel angle θ can be measured directly using the rotary encoder from the wheels.

As one can see, there is no way to get a direct measurement of the wheel velocity $\dot{\theta}$.

3.2.1 Angle Measurements

The gyro allows us to obtain the rotational rate of the MinSeg in three directions, where one of these correspond to the angular rate. However, since there will be calibration errors, the true angular rate can be written as:

$$\omega_m(t) = \dot{\alpha} + b$$

where b is the calibration error. The calibration error b was estimated in a simple automatic calibration process. Then the angular rate can be measured by simply subtracting this from the measurements from the gyrometer.

The angular rate can furthermore be used to get a measurement for the angle using that $\alpha(t) = \int_0^t \omega(\tau)d\tau + \alpha(0)$. Here, the $\alpha(0)$ angle is not known, so it is set to zero. Let the angle estimate from the gyro (α_g) denote the angle obtained by integrating the calibrated gyro measurement:

$$\alpha_g(t) = \int_0^t \omega(\tau) + b - \hat{b}d\tau = \alpha(t) - \alpha(0) + (b - \hat{b})t \tag{3}$$

This estimate α_g has multiple problems. To begin with, the initial angle $\alpha(0)$ is not known, and if the device is not calibrated perfectly there will be an error that grows with time. Thus this measurement can not be used by itself over a long period of time, however this estimate have good detection of when the angle changes.

The more direct way to obtain the measurements for the tilt angle is to use the accelerometer, which produces measurements $a_x(t), a_y(t), a_z(t)$. By simple geometry, the estimate of the tilt angle given by the accelerometer (α_a) can be measured as:

$$\alpha_a(t) = \arctan(\frac{a_y(t)}{a_z(t)}). \tag{4}$$

As the accelerometer will pick up lots of signals when the MinSeg is moving, this reasoning hold when it is still. This will of course not be the case, so this measurement cannot be fully trusted either.

However, we can combine the two estimates α_g and α_a to get something useful. As α_g works well on detecting changes, and α_a works well when the MinSeg is not moving to much, a high-pass filter can be used on α_g and a low-pass filter on α_a:

$$\alpha(s) = H(s)\alpha_g(s) + L(s)\alpha_a(s) = 1 - L(s)\alpha_g(s) + L(s)\alpha_a(s) \tag{5}$$

The low pass filter was a simple first order filter with bandwidth 3rad/s, $L(s) = \frac{1}{s/3+1}$, and the high pass filter was its complement $H(s) = 1 - L(s)$. After ZOH-sampling of (5), the following difference equation was obtained:

$$\alpha(k+1) = e^{-3h}\alpha(k) + (1 - e^{-3h})\alpha_a(k) + \alpha_g(k+1) - \alpha_g(k) \tag{6}$$

Although this procedure gave a decent result, it was found to work better together with a Kalman filter. More on this in the later section about the Kalman filters.

3.2.2 Wheel-angle and Velocity Measurements

The wheel-angle measurement θ is not very noisy, however quantized with a resolution of 0.5 degrees gives some inaccuracies.

There is no way to get a direct measurement of the velocity $\dot{\theta}$. On way to solve this by making an approximate differentiating of the wheel-angle. This was tried, however the results were not satisfactory. Instead a Kalman filter for the wheels angle and position were used, using only the measure from the rotational encoder as input, and giving improved measurements of both θ and $\dot{\theta}$

3.3 Controller parameters

3.3.1 State feedback

To stabilize the MinSeg, a state feedback controller was used. The idea is to chose the control signal according to $u = -Kx$, where the matrix K is chosen to obtain some desired characteristic of the system. The common way to do this is to choose a pole placement and find the parameters of K so that the closed loop system has the desired poles. In order to do this for any pole placement, the system must be controllable, but that is quickly verified by computing the controllability matrix

$$W_c = \begin{pmatrix} \Gamma & \Phi\Gamma & \Phi^2\Gamma & \Phi^3\Gamma \end{pmatrix} = \begin{pmatrix} -0.1930 & -0.0014 & -0.0020 & -0.0026 \\ -0.0017 & -0.0019 & -0.0020 & -0.0020 \\ 2.3497 & 0.0018 & 0.0007 & 0.0007 \\ 0.0205 & 0.0235 & 0.0235 & 0.0235 \end{pmatrix} \tag{7}$$

and noting that it has rank 4. From the initial testing in Simulink, putting the poles in -2, -4 and $3 \pm 2i$ seemed to work well. The corresponding feedback matrix then became $K = [-0.0855, -0.7922, -0.4257, -0.0018]$, and these parameters were used as an initial guess when testing the physical process. However, they were found to be quite far from optimal, so another set of parameters were found using LQR.

3.3.2 Linear Quadratic Regulator (LQR)

The LQR-algorithm is an automatic way of finding an appropriate state feedback controller. It is essentially an optimization problem, where the goal is to find a control law $u_k = -Kx_k$ which minimizes the cost function

$$J = \sum_{k=0}^{\infty} x_k^T Q x_k + r u_k^2. \tag{8}$$

To guarantee the existence of a solution, the matrix Q must be positive definite, and r must be positive. For simplicity, a diagonal matrix Q was used. The diagonal elements q_{ii} are chosen as the square reciprocals of the intended working range of the state x_i, and similarily r is the square reciprocal of the intended working range of the control signal. For instance, the tilt angle should preferrably not deviate more than one degree from the upright position, so $q_{22} = \left(\frac{180}{\pi}\right)^2$. The other values were chosen somewhat arbitrarily, but they were only intended to give a good first estimate of suitable controller parameters, which could be tweaked later. The new feedback matrix was $K = [-5.8141, -16.235, -0.4739, -0.0141]$.

3.4 Kalman filter

To further remove noise from the signals as well as to get an approximation of the wheel velocity $\dot{\theta}$, two Kalman filters were implemented. One Kalman filter estimates the tilt angle and velocity $(\alpha, \dot{\alpha})$ and the other one the wheel angle and velocity $(\theta, \dot{\theta})$ The parameters for the Kalman filters, matrices A_{tilt}, A_{wheel}, B_{tilt}, C_{tilt}, C_{wheel}, D_{tilt} and the vectors B_{wheel} and D_{wheel}, were provided by the project supervisor. In the control loop at an iteration k, the states are first estimated with the values measured during the same iteration, and the estimations from the previous one. To minimise output delay, the control signal is then calculated and applied. After that, the states for the next iteration are updated using the measurements and estimations from the current iteration. This two step process is done so that the estimated states can be updated with θ with the most recent measurements before being used to calculate the control signal, reducing latency between estimations and measurements. Note that for the wheel estimations, only the measured wheel angle is used which is why B_{wheel} and D_{wheel} are vectors instead of matrices.

$$\begin{pmatrix} \hat{\dot{\alpha}} \\ \hat{\alpha} \end{pmatrix}_{k|k} = C_{tilt} \begin{pmatrix} \hat{\dot{\alpha}} \\ \hat{\alpha} \end{pmatrix}_{k|k-1} + D_{tilt} \begin{pmatrix} \dot{\alpha} \\ \alpha \end{pmatrix}_k \tag{9}$$

$$\begin{pmatrix} \hat{\dot{\alpha}} \\ \hat{\alpha} \end{pmatrix}_{k+1|k} = A_{tilt} \begin{pmatrix} \hat{\dot{\alpha}} \\ \hat{\alpha} \end{pmatrix}_{k|k-1} + B_{tilt} \begin{pmatrix} \dot{\alpha} \\ \alpha \end{pmatrix}_k \tag{10}$$

$$\begin{pmatrix} \hat{\dot{\theta}} \\ \hat{\theta} \end{pmatrix}_{k|k} = C_{wheel} \begin{pmatrix} \hat{\dot{\theta}} \\ \hat{\theta} \end{pmatrix}_{k|k-1} + D_{wheel}\theta_k \tag{11}$$

$$\begin{pmatrix} \hat{\dot{\theta}} \\ \hat{\theta} \end{pmatrix}_{k+1|k} = A_{wheel} \begin{pmatrix} \hat{\dot{\theta}} \\ \hat{\theta} \end{pmatrix}_{k|k-1} + B_{wheel}\theta_k \tag{12}$$

The actual matrices used for the Kalman filters were the following for α and $\dot{\alpha}$

$$C_{tilt} = \begin{pmatrix} 0.3820 & -0.0015 \\ -0.0075 & 0.9911 \end{pmatrix}, \quad D_{tilt} = \begin{pmatrix} 0.6180 & 0.0015 \\ 0.0075 & 0.0089 \end{pmatrix}$$
$$A_{tilt} = \begin{pmatrix} 0.3820 & -0.0015 \\ 0.0001 & 0.9911 \end{pmatrix}, \quad B_{tilt} = \begin{pmatrix} 0.6180 & 0.0015 \\ 0.0199 & 0.0089 \end{pmatrix}$$

and for the wheel position θ and wheel velocity $\dot{\theta}$:

$$C_{tilt} = \begin{pmatrix} 1 & -9.90790 & 0.4618 \end{pmatrix}, \quad D_{tilt} = \begin{pmatrix} 9.9079 \\ 0.5382 \end{pmatrix}$$
$$A_{tilt} = \begin{pmatrix} 1 & -9.9079 \\ 0.0200 & 0.2636 \end{pmatrix}, \quad B_{tilt} = \begin{pmatrix} 9.9079 \\ 0.7364 \end{pmatrix}$$

4 Results

With the control algorithm described above, the MinSeg was able to balance in a controlled fashion indefinitely, see figure 3. That being said, the MinSeg does have to oscillate around the setpoint as it always has to stay in motion to balance. This can be seen in figure 2 and 4. In addition to this, the MinSeg managed to follow an alternating reference, moving back and forth. The MinSeg does still have certain restrictions though. The maximum speed that was achieved when updating the reference was π rad/s which is rather slow compared to the speed when full voltage is applied to the motor. This is illustrated in figure 4, as the control signal is below -2 V when the MinSeg is moving in time with the decreasing reference. The MinSeg does also experience a stationary error when following the setpoint, which can be seen in figure 4. An attempt to remove this was made by adding an integrator for the feedback of the wheel position, but this made it harder for it to follow a changing setpoint and was thus removed. Another disadvantage that is visible in figure 4 is that the resolution of the wheel position is rather low and the quantization is clear. This happens because there is a limit of 8-bits for communicating with the HC-06 Bluetooth module.

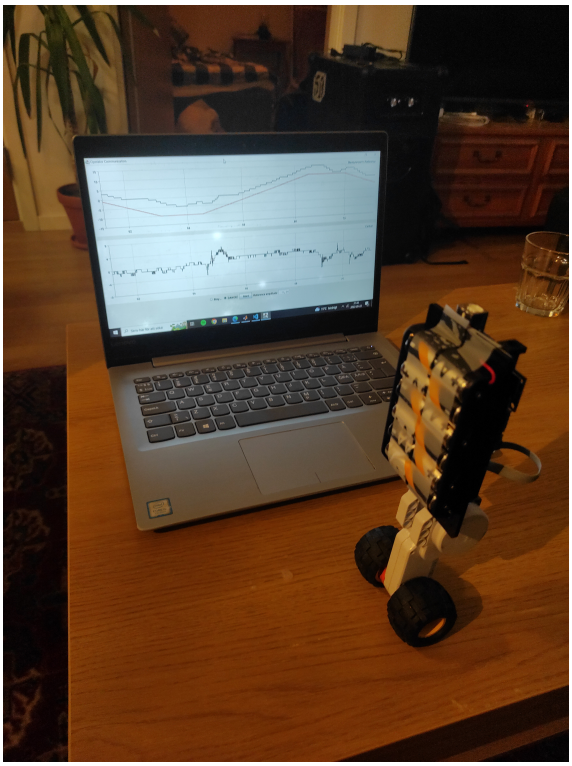

Figure 3: MinSeg balancing with GUI running in the background.

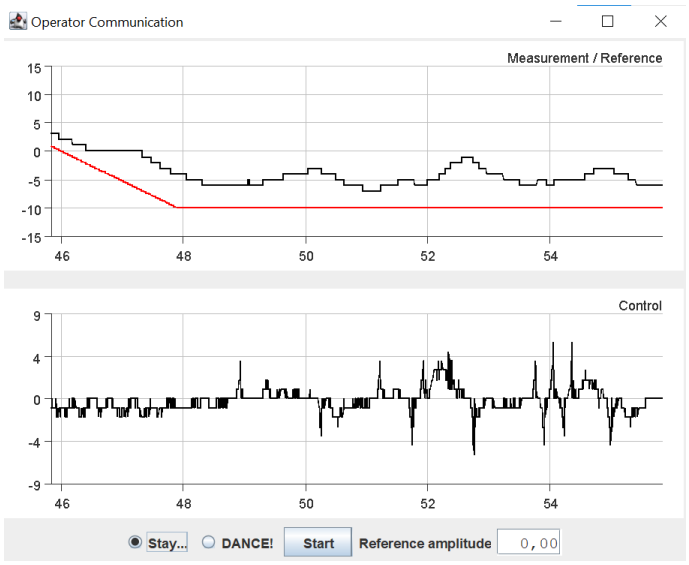

Figure 4: Measurements from the balancing MinSeg. The stationary error is clearly visible as the MinSeg is oscillating around -5 rad while the setpoint is at -10 rad.

5 Conclusion

With quite a few setbacks and problems along the way, the MinSeg is balancing and can move according to a reference. Throughout the project we have used and gotten familiar with Arduino IDE and how it can be used to program an Arduino board, specifically for control applications. To get the segway to balance the measurements from two sensors had to be understood and converted into something useful for a control law. Different approaches to the control principle was tested untill the process worked as expected. It is important to continue check the signals, because our main problem was with noise that was removed with a Kalman filter. On the Java side we have made our own GUI for our process using Java Swing, and setting up the communication between it and the arduino. This was done using a bluetooth module and a java library, something that could (with some effort) be converted into a phone app. To summarize, from this project we have learned how to take a control project from nothing but hardware to whatever our goal is.