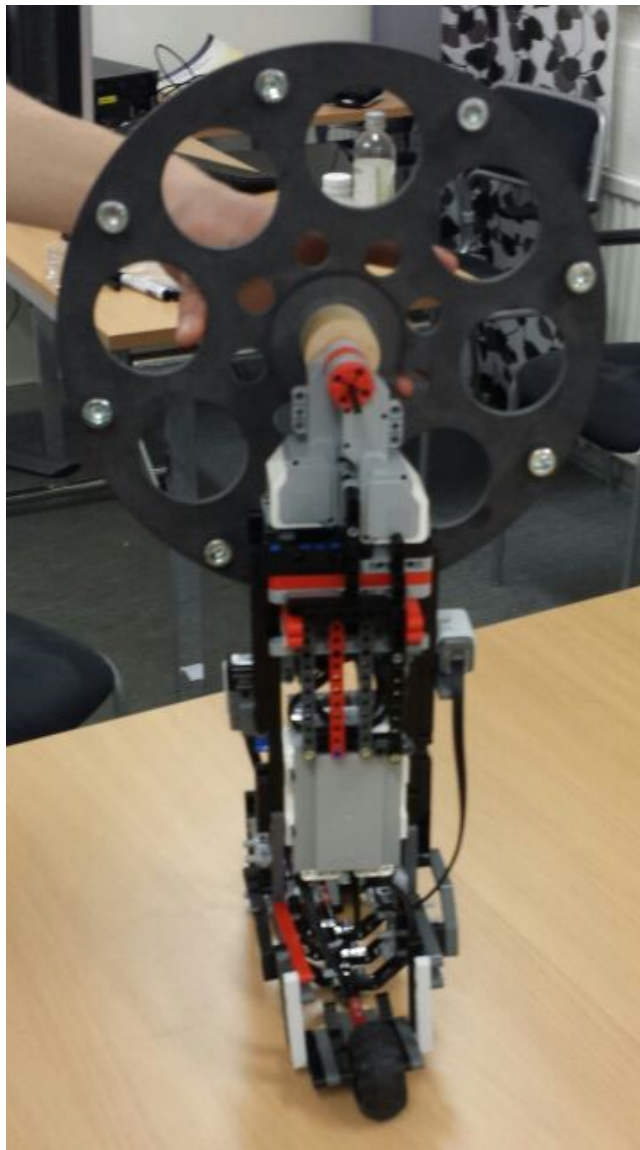


FRT090

Feedback Seminar 2

Group B

December 6, 2014



1 Introduction

We are designing a unicycle made by lego with an integrated inertia wheel to keep the lateral balance. The project therefore consists of two problems. To stabilize the unicycle in the medial direction by designing a controller for the ground wheel, and to stabilize it in the lateral direction by applying a reaction wheel connected to the top of the unicycle.

2 Design

2.1 Inertia wheel

We realized early in the project that controlling the inertia wheel is the most complicated part of the project. The complexity comes from several areas. The sensors have to be very accurate (which they are not), the simulation in simulink had to take account of the relationship between torque and the speed of the wheel among other things, we had to design a working controller.

The unicycle is similar to an inverted pendulum with a point mass in the center of mass, with an additional mass that is the inertia wheel, seen in figure 1. We modeled the pendulum as can be seen in (1). The relevant states from the model are the angle of the pendulum, the angular velocity of the pendulum and the angular velocity of the inertia wheel. The angular position of the inertia wheel is not important as only the change in angular position affects the torque. Torque is the momentum you get when a vector force is acted on the wheel with a certain radius. The torque acting on the pendulum due to gravity (which makes it fall) must be less than the torque from the inertia wheel (which we can control) for it to be stable. The reaction torque is therefore dependent on the radius of the wheel. The maximum torque of the engine is also limited, so some angles are impossible for the reaction wheel to stabilize.

To optimize the controller we implemented a linear quadratic regulator which minimizes the cost function, seen in (2). The Q matrix determines how much each state will be penalized. Since we want to keep cost function (J) small a large Q will require that the states are small. $Q = \begin{bmatrix} q_1 & 0 & 0 \\ 0 & q_2 & 0 \\ 0 & 0 & q_3 \end{bmatrix}$.

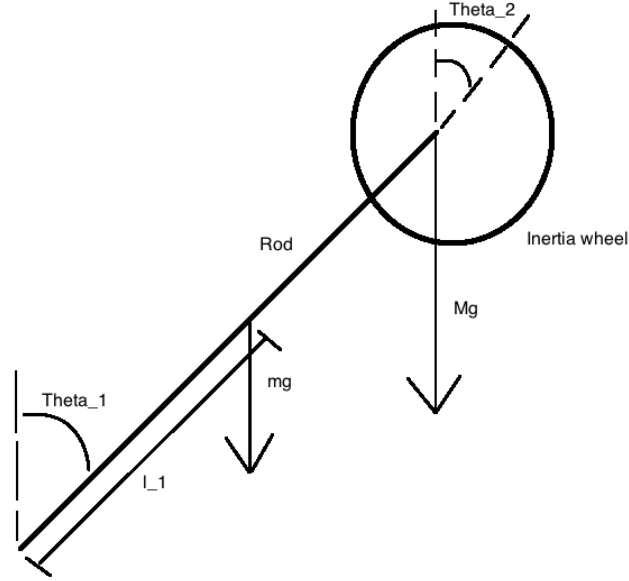


Figure 1: Shows the sketch of the inverted pendulum

$$\begin{aligned} (J_w + ML^2 + J_p) \frac{\partial^2 \theta_1}{\partial t^2} &= l_1 \sin \theta_1 mg + L \sin \theta_1 Mg - \tau \\ J_w \frac{\partial^2 \theta_2}{\partial t^2} &= \tau \end{aligned} \quad (1)$$

2.1.1 LQR design

The cost function we needed to minimize was:

$$E(x^T Qx + u^T Ru) \quad (2)$$

We set the weight matrices to the values:

$$q1 = 1/1.57$$

$$q2 = 1/0.349$$

$$q3 = 1/16$$

$$R = 1/0.348$$

These values come from the bounds of the states. The pendulum angle has a range of -1.57 to 1.57 radians (-90 to 90 degrees). We estimated that the maximum speed of the pendulum from free fall is 0.349 rad/s (about 20 degrees per second). The maximum angular velocity of the reaction wheel was determined to be 16 rad/s. Max torque is 0.348 Nm. These values are used to normalize each product of the cost function so they are approximately equal. These values may need to be tuned, but since the system is not adequately stable there are other problems we need to solve before increasing robustness

In simulink we had to take into account the actuator, being the motor. The torque of the motor rapidly decreases as the speed of the wheel increases. Therefore the max torque could only be achieved at a short time. After several attempts with a PID controller we concluded that with these limitations we had to use state feedback with the cost function shown above.

2.2 Implementation

For the state feedback implementation we used the floating point algorithm, we will not go in to details how the whole implementation process has been made but there are quite important steps in the implementing process which are worthy to be underlined.

To discretize the continuous parameters in our system we used the zero hold method.

One problematic issue was that when parameters (the estimated parameters in the model) are changed due to the testing experiments when the model parameters are changed that is to say the new A and B matrices are dependent on the sampling time (h). The sampling time is still a issue and have not been resolved but are planned to be.

Moving on to the programs we use and why we choose to work with them. The operating system we choose to work with is the Windows 7 because of the simplicity compared to working with the Linux operating system. Next we have the main programming language which we choose to be Lejos which is an extension of java and incorporates all that is Java and its benefits. We are all in the group familiar with the java language and therefore this was the language for us. The GUI that we write the java programs is in Eclipse due to the familiar appearance of it and functionality. With Eclipse we can run programs directly in the ev3. However sending data from the ev3 to the computer is managed with the free program which is Winscp (a client server program) that allows us to send data from the ev3 to the computer via bluetooth and also comes with a easy to work with GUI (`GraphicUserInterface`).

The format we use to send data from the ev3 to the computer is "txt" which can easily be managed with matlab which allow us to plot the data directly in matlab. However we can not observe the data in realtime. Furthermore we have been experiencing slow cycles in the main loop of our program which is the discretization of the state feedback. Therefore we are now saving all our data in multiple arrays and after the test run the program writes the data from the arrays to the txt file to be further send to the computer for further processing.

Furthermore we use state feedback which means that we need to measure the three states that we have which in this case are the angle θ , angle velocity $\dot{\theta}$ and the velocity of the inertia wheel.

To obtain the angle we use two sensors, the gyro and the accelerometer sensor. In this case we are using the "NXT Gyro Sensor for LEGO Mindstorms NXT" respectively HiTechnic NXT AccelerationSensor for LEGO Mindstorms NXT. The gyro sensor gives us the change in radians which we integrate in our code while the accelerometer gives us the angle directly. Because of the nature of the accelerometer which is that it is very noisy (does not give a stable signal) we incorporate the pyrometer as well. However the gyro sensor has also flaws which is that after some time the angle of the gyro sensor increases from the actual angle because of the integration. Therefore we use only want the low frequency of

the accelerometer and high frequency of the gyroscope so we use low pass filter for the accelerometer and a high pass filter for the gyro sensor. This method of using the two sensors together complementary filter.

To obtain the change in angle we just use the gyroscopes measurement while to obtain the speed of the inertia wheel we use incorporated methods given in Lejos to get the amount of change in radians and divide with the cycle of the main loop (the time between measurement of the change in radians).