Self-Balancing Robot

CSE 499 Embedded Controls Group Project

Andrew Woska, Daniel Maas, Imani Muhammad-Graham

Spring 2022

Table of Contents

1 System Description

The system is a self balancing robot. It takes input from the robot's angular acceleration that is used, in conjunction with the motors attached to the wheels, to create acceleration in the opposite direction to counteract the falling motion. The initial design used tilt sensors as the primary input instead of the innate acceleration of the wheels, however, this idea was changed because the sensors available were not able to provide accurate tilt readings. All of the input and output peripherals in this design are regulated by a Nucleo L4R5ZI microprocessor that is using code developed in the Mbed programming environment. The programming language used is the standard C++ programming language. All of the aforementioned devices are housed inside a plastic chassis from a former robot.

2 System Specification

Inputs:

- Accelerometer
- Tilt Sensors

Outputs:

- DC Motor for robot

Constraints:

- ADC and I2C utilizations by the accelerometer
- Tilt sensor has limited angle representation
- Uses timing elements in code
- MIT license used in accelerometer driver must be respected

3 System Development

3.1 System Functions

The following functions are utilized to create this system:

3.1.1 Motor Power based on Inverse Pendulum

$$\frac{\Phi(s)}{U(s)} = \frac{\frac{ml}{q}s}{s^3+\frac{b(I+ml^2)}{q}s^2-\frac{(M+m)mgl}{q}s-\frac{bmgl}{q}}$$

3.1.2 Robot Angle

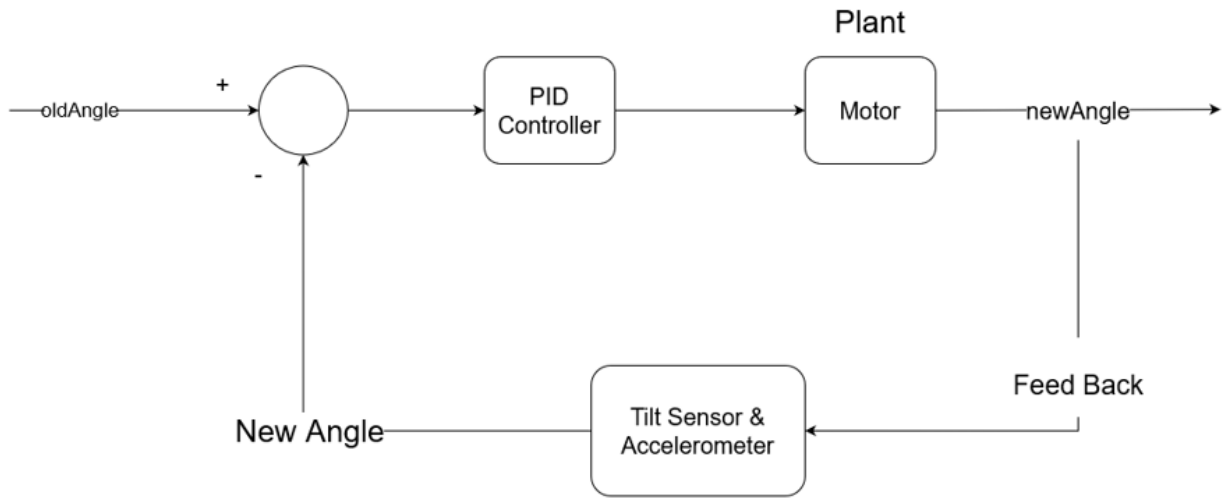
$$\phi = tan^{-1} \frac{accel\,Y}{accel\,Z} \cdot \frac{180}{\pi}$$

3.1.3 PID Controller

$$PID = Kp \cdot e(t) + Ki \int e(t)dt + Kd \cdot \frac{de}{dt}$$

3.2 Control System

The control system can be modeled as:

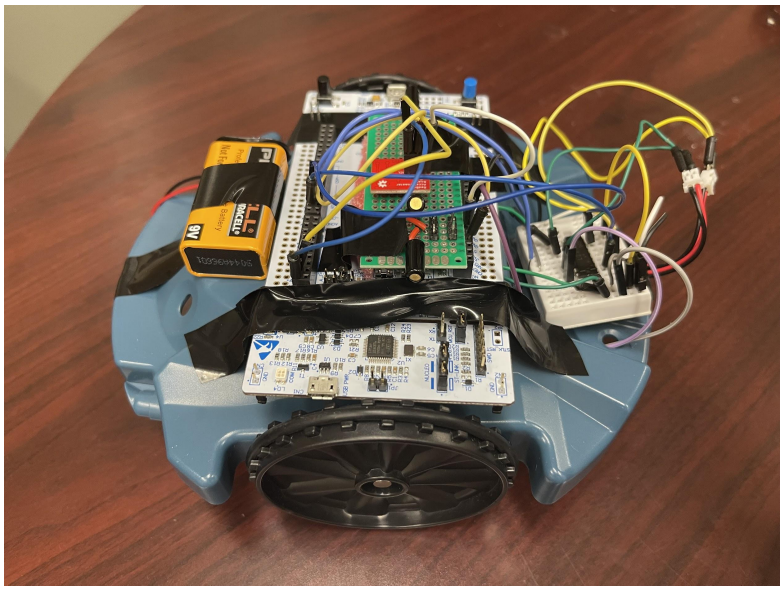


Using a MATLAB simulation, we can determine the values of Kp, Ki, and Kd from the formula in section 3.1.3. The simulation results show that the following values should be sufficiently stable for the system:

- Kp = 10
- Ki = 7
- Kd = .75

The simulations are under Appendix A.

3.3 Building the System



The system's design can be separated into two parts: hardware and software. Both parts are strongly coupled, as the control system is based on the hardware used. The following materials and tools were used to build the system:

Bill of Materials:

- Plastic car
- 2 DC motors with wheels
- RedBot accelerometer
- 4 tilt sensors
- 9V Battery
- Power Module
- Nucleo L4R5ZI
- Mbed OS 6

Hardware:

- Mounted tilt sensors and the accelerometer in the center of mass for accurate angle measurements
- Modified existing robot car to fit Nucleo microcontroller, sensors, motors, power supply, and

allow room to tilt

Software:

- Programmed on Mbed OS to receive signals from sensors and send signals to motor

4 Embedded and RTOS Elements

4.1 Embedded Elements

Two elements make up the embedded systems element of the system: ADC and memory access. The ADC or analog-to-digital converter takes time in order to process. There must be a sufficient amount of time for the ADC to process. In order to protect from memory corruption and leakage, a mutex protects the data from change unless the mutex is locked.

4.2 RTOS elements

Two elements make up the RTOS elements of the system: watchdog and time. The watchdog prevents the system from malfunctioning if the deadline is not met. This ensures the system continues working even in the event of crashing. The system makes the main thread sleep to process changes and update the system.

5 Functions

5.1 setup

This function sets up all the sensors, starts the watchdog, and initializes values.

5.2 controller

This function uses the PID controller to control the motor.

5.3 setMotors

This function controls the direction and speed of the motors. It could go forwards, backwards, and stop.

5.4 calcRoll

This function calculates the roll of the system. Roll is one of the 6 degrees-of-freedom that an object can have. It outputs the angle in degrees.

6 Analysis

6.1 System Outcomes

Although the system works, there are some concerns regarding its efficiency and accuracy.

6.2 System Improvements

Future cases to address are as follows:

Accuracy:

- Mount tilt sensors in a more effective position so they can work as intended

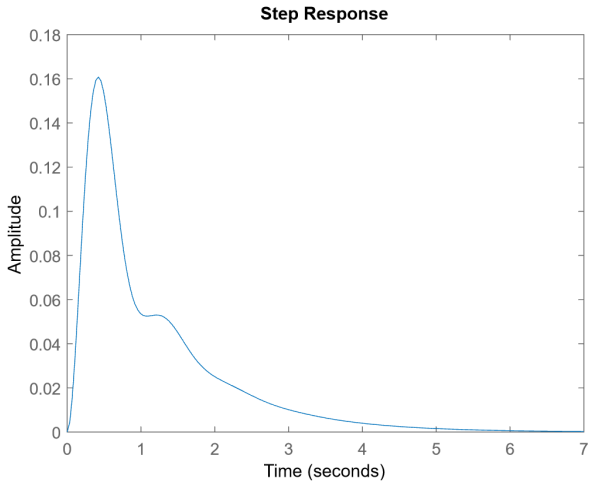
Software/Efficiency

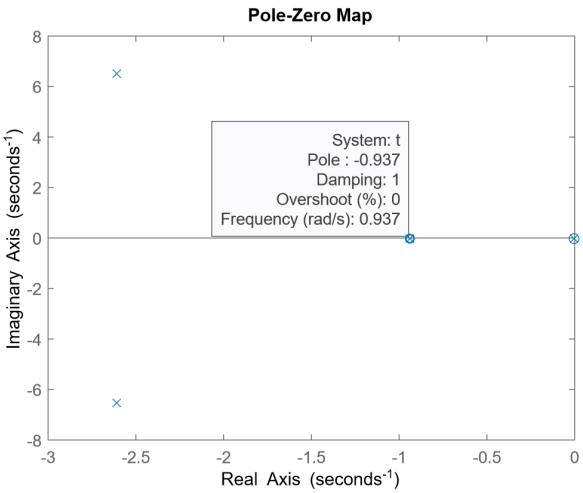
- Collect data to determine optimal sample rate
- Analyze stability in order to improve functionality of controller
- Account for additional edge cases, i.e. car falls over

Motor

- Research H-bridge malfunction; fix motor to rotate at various speeds

Appendix A: MATLAB Simulations





Appendix B: MATLAB Code

```
% @file cse499_proj.m

% @author Andrew Woska, Daniel Maas, Imani Muhammed-Graham

% @date 2022-03-14

% for CSE 499 - Embedded Controls

% Description

%   This file is used to calculate and test the PID controller

%   for a self-balancing robot


clc

clear

close all


M = 0.2;

m = 0.1;

b = 0.1;

I = 0.006;

g = 9.8;
```

```
l = 0.2;
q = (M+m)*(I+m*l^2)-(m*l)^2;

num = [m*l/q, 0];
denom = [ 1, b*(I+m*l^2)/q, -(M+m)*m*g*l/q, -b*m*g*l/q ];

% s = .001; % sample timing
% PID values
Kp = 10;
Ki = 7;
Kd = .75;
% controller
sys = tf(num,denom)
c = pid(Kp,Ki,Kd)
t = feedback(sys,c)
% create figures
figure
pzmap(t)
figure
step(t)
```