معهد الأهرامات العالي للهندسة والتكنولوجيا
P.H.I For Engineering and Technology

DIGITAL CONTROL PROJECT

(Balancing Robot)

DR / GAMAL AL SHEKH

ENG /AHMED EMAD

Mohamed Ahmed Awwad	**20130137**
Abdullah Sobhy Hamed	**20130419**
Mohamed Wael Wagdy	**20130189**
Hazem Adel Ahmed	**20130344**
Mostafa Ahmed Saied	**20140432**

Contents

1. Abstract

Self-balancing robot is based on the principle of Inverted pendulum, which is a two wheel vehicle balances itself up in the vertical position with reference to the ground. It consist both hardware and software implementation.

Mechanical model based on the state space design of the cart, pendulum system. To find its stable inverted position, I used a generic feedback controller (i.e. PID controller).

According to the situation we have to control both angel of pendulum and position of cart. Mechanical design consist of two dc gear motor with encoder, one Arduino microcontroller, IMU (inertial mass unit) sensor and motor driver as a basic need.

IMU sensor which consists of accelerometer and gyroscope gives the reference acceleration and angle with respect to ground (vertical), when encoder which is attached with the motor gives the speed of the motor.

These parameters are taken as the system parameter and determine the external force needed to balance the robot up. It will be prevented from falling by giving acceleration to the wheels according to its inclination from the vertical. If the bot gets tilts by an angle, than in the frame of the wheels; the center of mass of the bot will experience a pseudo force which will apply a torque opposite to the direction of tilt.

2. Introduction

To make a self-balancing robot, it is essential to solve the inverted pendulum problem or an inverted pendulum on cart. While the calculation and expressions are very complex, the goal is quite simple: the goal of the project is to adjust the wheels' position so that the inclination angle remains stable within a pre-determined value (e.g. the angle when the robot is not outside the premeasured angel boundary).

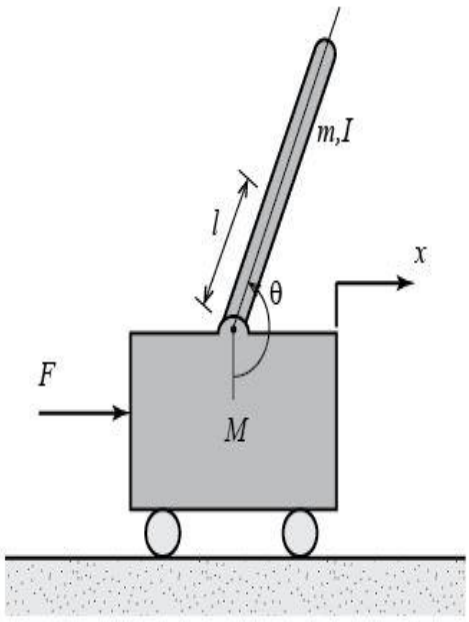
When the robot starts to fall in one direction, the wheels should move in the inclined direction with a speed proportional to angle and acceleration of falling to correct the inclination angle.

So I get an idea that when the deviation from equilibrium is small, we should move "gently" and when the deviation is large we should move more quickly. The Inverted pendulum is a highly unstable and non-linear system; it's one of the complex systems to be controlled in control engineering due to its non-linear dynamics. Inverted pendulum system is studied in detail. Following is the model of an inverted pendulum system with cart.

To simplify things a little bit, I take a simple assumption; the robot's movement should be confined on one axis (e.g. only move forward and backward) and thus both wheels will move at the same speed in the same direction. Under this assumption the mathematics become much simpler as we only need to worry about sensor readings on a single plane.

If we want to allow the robot to move sidewise, then you will have to control each wheel independently. The general idea remains the same with a less complexity since the falling direction of the robot is still restricted to a single axis. The modeling of the dynamics of the physical system is needed to understand its behavior

3. System modeling



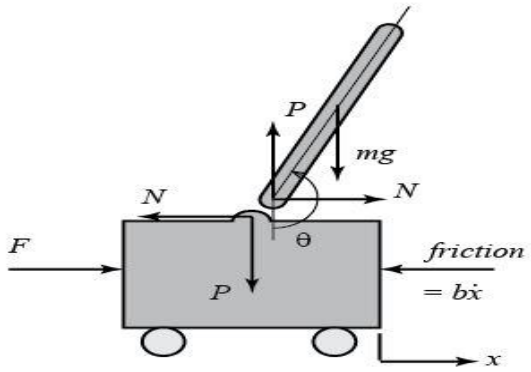
The system in this example consists of an inverted pendulum mounted to a motorized cart. The inverted pendulum system is an example commonly found in control system textbooks and research literature. Its popularity derives in part from the fact that it is unstable without control, that is, the pendulum will simply fall over if the cart isn't moved to balance it. Additionally, the dynamics of the system are nonlinear. The objective of the control system is to balance the inverted pendulum by applying a force to the cart that the pendulum is attached to. A real-world example that relates directly to this inverted pendulum system is the attitude control of a booster rocket at takeoff. In this case we will consider a two-dimensional problem where the pendulum is constrained to move in the vertical plane shown in the figure below. For this system, the control input is the force F that moves the cart horizontally and the outputs are the angular position of the pendulum θ and the horizontal position of 0the cart X

For this example, let's assume the following quantities:

(M)	Mass of the cart	0.5 kg
(m)	Mass of the pendulum	0.2 kg
(b)	coefficient of friction for cart	0.1 N/m/sec
(l)	Length to pendulum center of mass	0.3 m
(I)	mass moment of inertia of the pendulum	0.006 kg.m^2
(F)	Force applied to the cart	
(x)	Cart position coordinate	
(Theta)	pendulum angle from vertical (down)	

Force analysis and system equations

Below are the free-body diagrams of the two elements of the inverted pendulum system.



Summing the forces in the free-body diagram of the cart in the horizontal direction, you get the following equation of motion.

$$M\ddot{x} + b\dot{x} + N = F$$

Note that you can also sum the forces in the vertical direction for the cart, but no useful information would be gained. Summing the forces in the free-body diagram of the pendulum in the horizontal direction, you get the following expression for the reaction force N

$$N = m\ddot{x} + ml\ddot{\theta}\cos\theta - ml\dot{\theta}^2\sin\theta$$

If you substitute this equation into the first equation, you get one of the two governing equations for this system.

$$(M+m)\ddot{x} + b\dot{x} + ml\ddot{\theta}\cos\theta - ml\dot{\theta}^2\sin\theta = F$$

To get the second equation of motion for this system, sum the forces perpendicular to the pendulum. Solving the system along this axis greatly simplifies the mathematics. You should get the following equation.

$$P\sin\theta + N\cos\theta - mg\sin\theta = ml\ddot{\theta} + m\ddot{x}\cos\theta$$

To get rid of the P and N terms in the equation above, sum the moments about the centroid of the pendulum to get the following equation.

$$-Pl\sin\theta - Nl\cos\theta = I\ddot{\theta}$$

Combining these last two expressions, you get the second governing equation

$$(I + ml^2)\ddot{\theta} + mgl\sin\theta = -ml\ddot{x}\cos\theta$$

Since the analysis and control design techniques we will be employing in this example apply only to linear systems, this set of equations needs to be linearized. Specifically, we will linearize the equations about the vertically upward equillibrium position, $\theta = \pi$, and will assume that the system stays within a small neighborhood of this equilibrium. This assumption should be reasonably valid since under control we desire that the pendulum not deviate more than 20 degrees from the vertically upward position. Let ϕ represent the deviation of the pendulum's position from equilibrium, that is, $\theta = \pi + \phi$. Again presuming a small deviation (ϕ) from equilibrium, we can use the following small angle approximations of the nonlinear functions in our system equations:

$$\cos\theta = \cos(\pi + \phi) \approx -1$$

$$\sin\theta = \sin(\pi + \phi) \approx -\phi$$

$$\dot{\theta}^2 = \dot{\phi}^2 \approx 0$$

After substiting the above approximations into our nonlinear governing equations, we arrive at the two linearized equations of motion. Note u has been substituted for the input F

$$(I + ml^2)\ddot{\phi} - mgl\phi = ml\ddot{x}$$

$$(M + m)\ddot{x} + b\dot{x} - ml\ddot{\phi} = u$$

1. Transfer Function

To obtain the transfer functions of the linearized system equations, we must first take the Laplace transform of the system equations assuming zero initial conditions. The resulting Laplace transforms are shown below.

$$(I+ml^2)\Phi(s)s^2 - mgl\Phi(s) = mlX(s)s^2$$

$$(M+m)X(s)s^2 + bX(s)s - ml\Phi(s)s^2 = U(s)$$

Recall that a transfer function represents the relationship between a single input and a single output at a time. To find our first transfer function for the output $\Phi(s)$ and an input of $U(s)$ we need to eliminate $X(s)$ from the above equations. Solve the first equation for $X(s)$.

$$X(s) = \left[\frac{I+ml^2}{ml} - \frac{g}{s^2}\right]\Phi(s)$$

Then substitute the above into the second equation.

$$(M+m)\left[\frac{I+ml^2}{ml} - \frac{g}{s^2}\right]\Phi(s)s^2 + b\left[\frac{I+ml^2}{ml} - \frac{g}{s^2}\right]\Phi(s)s - ml\Phi(s)s^2 = U(s)$$

0Rearranging, the transfer function is then the following

$$\frac{\Phi(s)}{U(s)} = \frac{\frac{ml}{q}s^2}{s^4 + \frac{b(I+ml^2)}{q}s^3 - \frac{(M+m)mgl}{q}s^2 - \frac{bmgl}{q}s}$$

where,

$$q = [(M+m)(I+ml^2) - (ml)^2]$$

From the transfer function above it can be seen that there is both a pole and a zero at the origin. These can be canceled and the transfer function becomes the following.

$$P_{pend}(s) = \frac{\Phi(s)}{U(s)} = \frac{\frac{ml}{q}s}{s^3 + \frac{b(I+ml^2)}{q}s^2 - \frac{(M+m)mgl}{q}s - \frac{bmgl}{q}} \qquad [\frac{rad}{N}]$$

Second, the transfer function with the cart position $X(s)$ as the output can be derived in a similar manner to arrive at the following.

$$P_{cart}(s) = \frac{X(s)}{U(s)} = \frac{\frac{(I+ml^2)s^2 - gml}{q}}{s^4 + \frac{b(I+ml^2)}{q}s^3 - \frac{(M+m)mgl}{q}s^2 - \frac{bmgl}{q}s} \qquad [\frac{m}{N}]$$

- **MATLAB**

We can represent the transfer functions derived above for the inverted pendulum system within MATLAB employing the following commands. Note that you can give names to the outputs (and inputs) to differentiate between the cart's position and the pendulum's position. Running this code in the command window produces the output shown below.

```
M = 0.5;

m = 0.2;

b = 0.1;

I = 0.006;

g = 9.8;

l = 0.3;

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

s = tf('s');

P_cart = (((I+m*l^2)/q)*s^2 - (m*g*l/q))/(s^4 + (b*(I + m*l^2))*s^3/q - ((M + m)*m*g*l)*s^2/q - b*m*g*l*s/q);
```

```
P_pend = (m*l*s/q)/(s^3 + (b*(I + m*l^2))*s^2/q - ((M + m)*m*g*l)*s/q - b*m*g*l/q);

sys_tf = [P_cart ; P_pend];

inputs = {'u'};
outputs = {'x'; 'phi'};

set(sys_tf,'InputName',inputs)
set(sys_tf,'OutputName',outputs)

sys_tf
```

2. State-Space

The linearized equations of motion from above can also be represented in state-space form if they are rearranged into a series of first order differential equations. Since the equations are linear, they can then be put into the standard matrix form shown below.

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\phi} \\ \ddot{\phi} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & \frac{-(I+ml^2)b}{I(M+m)+Mml^2} & \frac{m^2gl^2}{I(M+m)+Mml^2} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{-mlb}{I(M+m)+Mml^2} & \frac{mgl(M+m)}{I(M+m)+Mml^2} & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \phi \\ \dot{\phi} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{I+ml^2}{I(M+m)+Mml^2} \\ 0 \\ \frac{ml}{I(M+m)+Mml^2} \end{bmatrix} u$$

$$\mathbf{y} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \phi \\ \dot{\phi} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} u$$

MATLAB

```
M = .5;

m = 0.2;

b = 0.1;

I = 0.006;

g = 9.8;

l = 0.3;
```

```
p = I*(M+m)+M*m*l^2; %denominator for the A and B matrices

A = [0      1              0           0;
     0 -(I+m*l^2)*b/p  (m^2*g*l^2)/p   0;
     0      0              0           1;
     0 -(m*l*b)/p       m*g*l*(M+m)/p  0];
B = [     0;
     (I+m*l^2)/p;
          0;
        m*l/p];
C = [1 0 0 0;
     0 0 1 0];
D = [0;
     0];

states = {'x' 'x_dot' 'phi' 'phi_dot'};
inputs = {'u'};
outputs = {'x'; 'phi'};

sys_ss = ss(A,B,C,D,'statename',states,'inputname',inputs,'outputname',outputs)
```

4.PID CONTROLER

For the PID, frequency response and root locus sections of this problem, we are interested in the control of the pendulum angel from vertical and pendulum's position. Using transfer function which is best-suited for single-input, single-output (SISO) systems you can't control both the system output. Therefore, the design criteria deal with the cart's position and pendulum angle simultaneously.

We can, however, assume the controller's effect on the cart's position after the controller has been designed by hit and trial method. In the next sections, we will design a controller to keep the pendulum to a vertically upward position when it will undergo a sudden force.

Specifically, the design criteria are that the pendulum restores its upright position within 2 seconds and that the pendulum never inclined more than 0.08 radians away from vertical (that is controllable) after being disturbed by a force of magnitude 2 N. The pendulum will initially begin from any controllable position but employing state-space design techniques, we are more confined and eager to choose a multi-output system.

In our case, the inverted_ pendulum system on a cart is single-input (only external), multi-output (SIMO). Therefore, for the state-space model of the Inverted Pendulum on a cart, we will control both the cart's position and pendulum's angle. To make the design more accurate and well defined in this section, we will check the parameters a 0.2-meter step in the cart's desired position. Under these conditions, it is expected that the cart achieve its stable position within 5 seconds and have a rise time under 0.2 seconds. It is also desired that the pendulum archive to its vertical position in under 2 seconds, and further, that the pendulum angle not travel more than 30 degrees (0.35 radians) way from the vertically upward.

we will design a PID controller for the inverted pendulum system. In the design process we will assume a single-input, single-output plant as described by the following transfer function. Otherwise stated, we will attempt to control the pendulum's angle without regard for the cart's position.

$$P_{pend}(s) = \frac{\Phi(s)}{U(s)} = \frac{\frac{ml}{q}s}{s^3 + \frac{b(I+ml^2)}{q}s^2 - \frac{(M+m)mgl}{q}s - \frac{bmgl}{q}} \qquad [\frac{rad}{N}]$$

$$q = (M+m)(I+ml^2) - (ml)^2$$

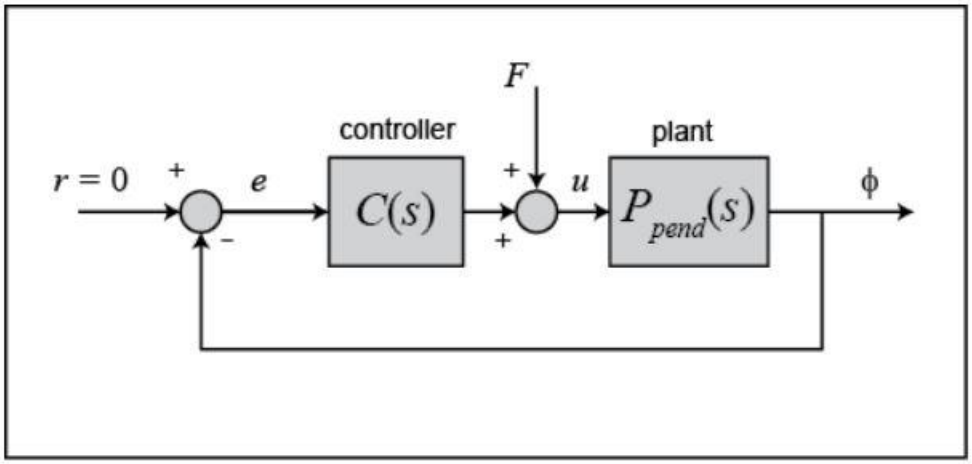
More specifically, the controller will attempt to maintain the pendulum vertically upward when the cart is subjected to a 1-Nsec impulse. Under these conditions, the design criteria are:

- Settling time of less than 5 seconds
- Pendulum should not move more than 0.05 radians away from the vertical For the original problem setup and the derivation of the above transfer function

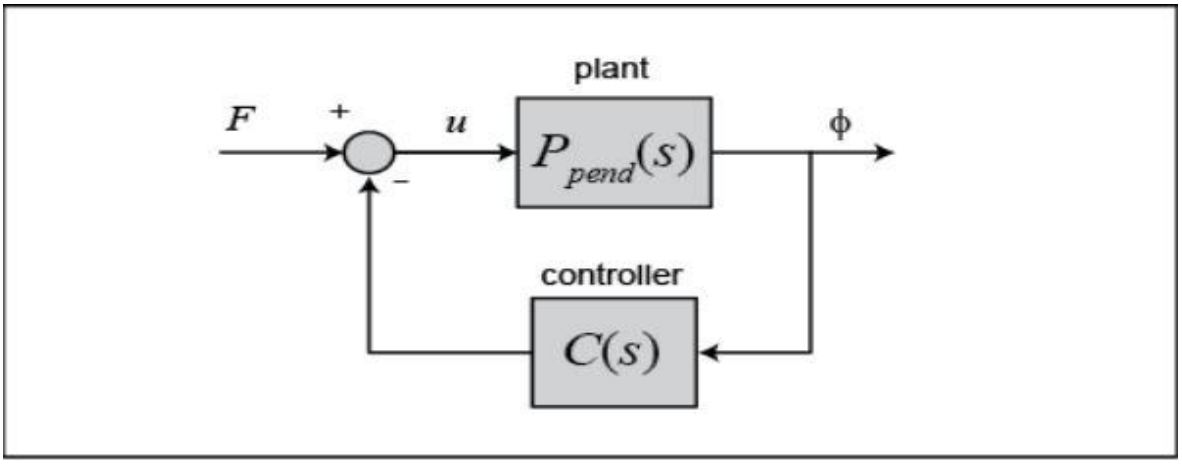
System structure

The structure of the controller for this problem is a little different than the standard control problems you may be used to. Since we are attempting to control the pendulum's position, which should return to the vertical after the initial disturbance, the reference signal we are tracking should be zero. This

type of situation is often referred to as a Regulator problem. The external force applied to the cart can be considered as an impulsive disturbance. The schematic for this problem is depicted below.



You may find it easier to analyze and design for this system if we first rearrange the schematic as follows.



The resulting transfer function for the closed-loop system from an input of force to an output of pendulum angle is then determined to be the following.

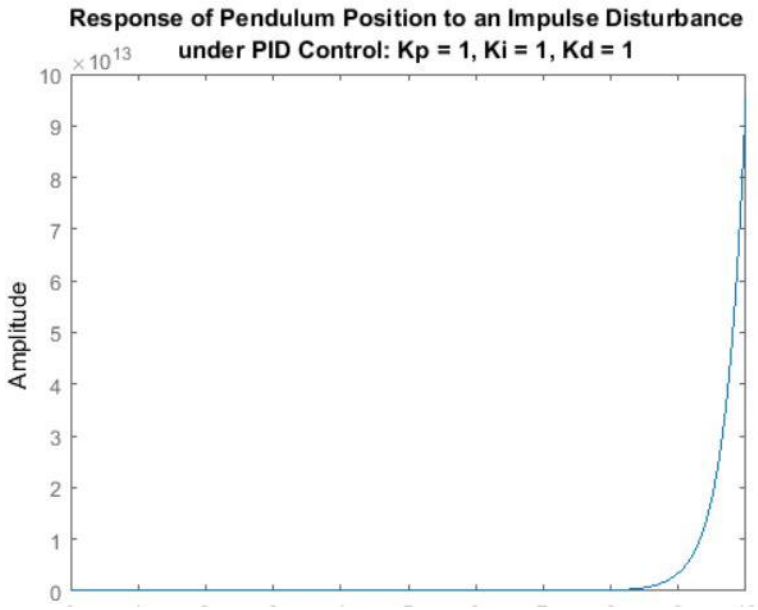
Before we begin designing our PID controller, we first need to define our plant within MATLAB. Create a new m-file and type in the following commands to create the plant model (refer to the main problem for the details of getting these commands).

```
M = 0.5;
m = 0.2;
b = 0.1;
I = 0.006;
g = 9.8;
l = 0.3;
q = (M+m)*(I+m*l^2)-(m*l)^2;
s = tf('s');
P_pend = (m*l*s/q)/(s^3 + (b*(I + m*l^2))*s^2/q - ((M + m)*m*g*l)*s/q - b*m*g*l/q);
```

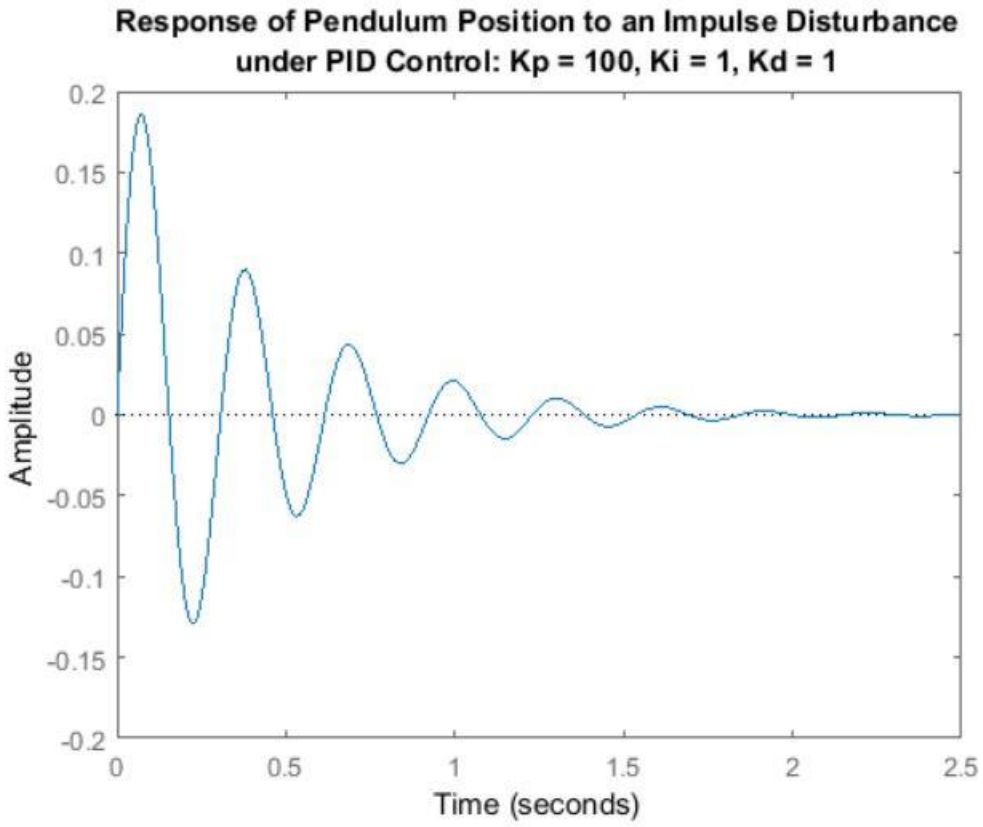
PID control

This closed-loop transfer function can be modeled in MATLAB by copying the following code to the end of your m-file (whether you're using the transfer function form or the state-space representation of the plant). Specifically, we define our controller using the PID object within,MATLABWe.then use the feedback command to generate the closed-loop transfer function as depicted in the figure above where the disturbance force is the input and the deviation of the pendulum angle from the vertical is the output.

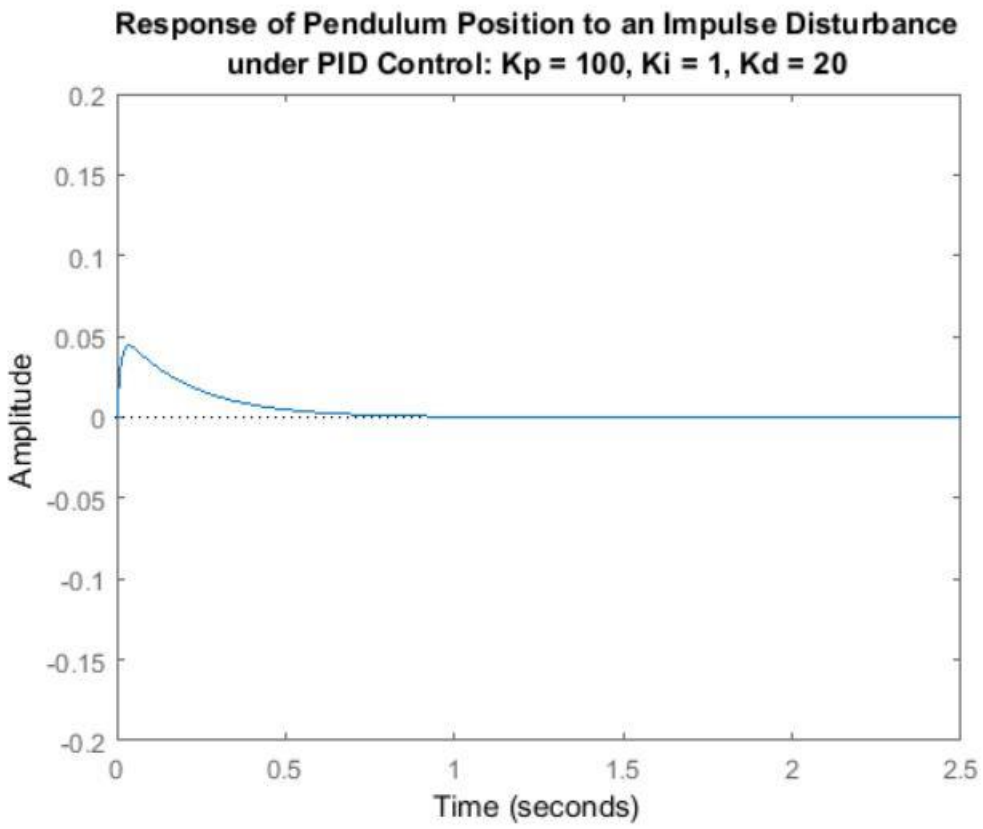
```
Kp = 1;

Ki = 1;

Kd = 1;

C = pid(Kp,Ki,Kd);

T = feedback(P_pend,C);
```



```
Kp = 100;

Ki = 1;

Kd = 1;

C = pid(Kp,Ki,Kd);

T = feedback(P_pend,C);

t=0:0.01:10;

impulse(T,t)

axis([0, 2.5, -0.2, 0.2]);

title({'Response of Pendulum Position to an Impulse Disturbance';'under PID Control: Kp = 100, Ki = 1, Kd = 1'});
```

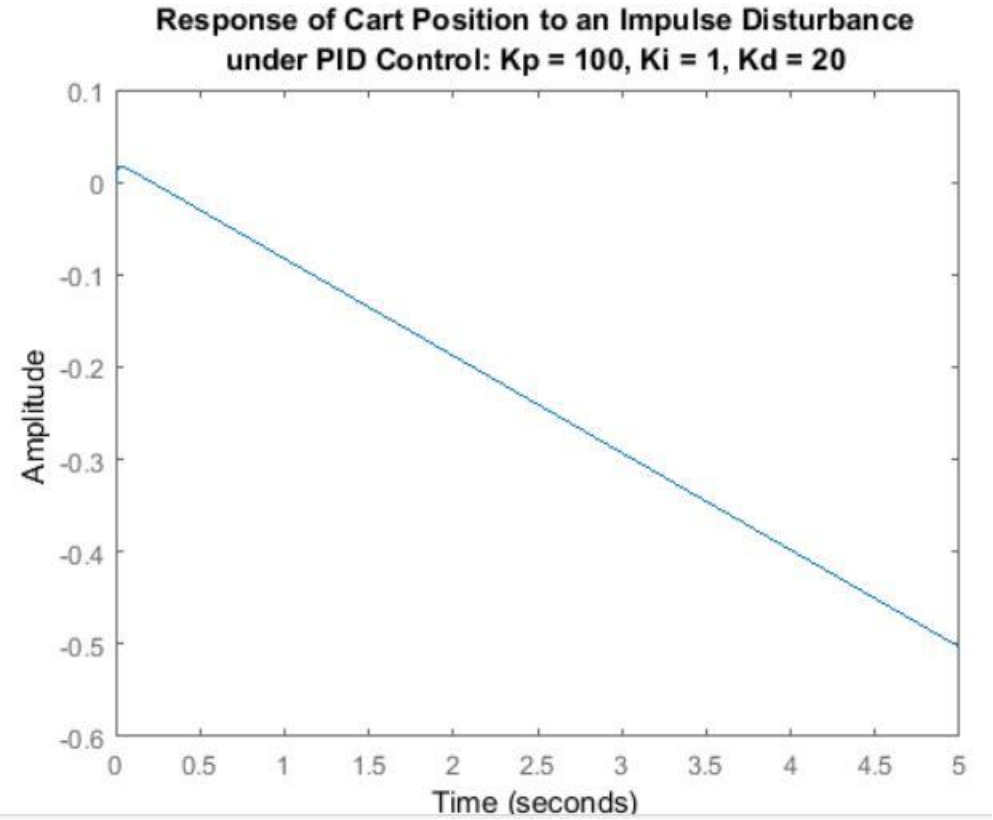


```
Kp = 100;

Ki = 1;

Kd = 20;

C = pid(Kp,Ki,Kd);

T = feedback(P_pend,C);

t=0:0.01:10;

impulse(T,t)

axis([0, 2.5, -0.2, 0.2]);

title({'Response of Pendulum Position to an Impulse Disturbance';'under PID Control: Kp = 100, Ki = 1, Kd = 20'});
```



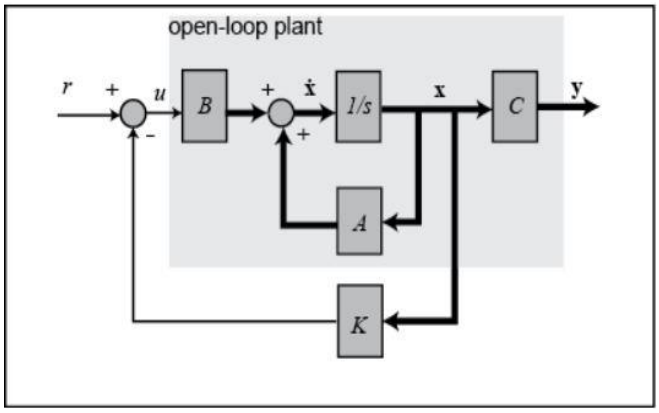
Resaponse of cart postion to an impulse disturbance under PID control

```
P_cart = (((I+m*l^2)/q)*s^2 - (m*g*l/q))/(s^4 + (b*(I + m*l^2))*s^3/q - ((M + m)*m*g*l)*s^2/q - b*m*g*l*s/q);

T2 = feedback(1,P_pend*C)*P_cart;

t = 0:0.01:5;

impulse(T2, t);

title({'Response of Cart Position to an Impulse Disturbance';'under PID Control: Kp = 100, Ki = 1, Kd = 20'});
```



5.STATE-SPACE

As you may have noticed if you went through some of the other inverted pendulum examples, the design criteria for this example are different. In the other examples we were attemping to keep the pendulum vertical in response to an impulsive disturbance force applied to the cart. We did not attempt to control the cart's position. In this example, we are attempting to keep the pendulum vertical while controlling the cart's position to move 0.2 meters to the right. A state-space design approach is well suited to the control of multiple outputs as we have here.



This problem can be solved using full-state feedback. The schematic of this type of control system is shown below where K is a matrix of control gains. Note that here we feedback all of the system's states, rather than using the system's outputs for feedback.

Open-loop poles

In this problem, r represents the step command of the cart's position. The 4 states represent the position and velocity of the cart and the angle and angular velocity of the pendulum. The output $\mathbf{y}$ contains both the position of the cart and the angle of the pendulum. We want to design a controller so that when a step reference is given to the system, the pendulum should be displaced, but eventually return to zero (i.e. vertical) and the cart should move to its new commanded position. To view the system's open-loop response please refer to the Inverted Pendulum: System Analysis page.

The first step in designing a full-state feedback controller is to determine the open-loop poles of the system. Enter the following lines of code into an m-file. After execution in the MATLAB command window, the output will list the open-loop poles (eigenvalues of A) as shown below.

```
M = 0.5;
```

```
m = 0.2;

b = 0.1;

I = 0.006;

g = 9.8;

l = 0.3;



p = I*(M+m)+M*m*l^2; %denominator for the A and B matrices

A = [0      1              0           0;
     0 -(I+m*l^2)*b/p   (m^2*g*l^2)/p   0;
     0      0              0           1;
     0 -(m*l*b)/p        m*g*l*(M+m)/p  0];
B = [     0;
     (I+m*l^2)/p;
          0;
        m*l/p];
C = [1 0 0 0;
     0 0 1 0];
D = [0;
     0];

states = {'x' 'x_dot' 'phi' 'phi_dot'};
inputs = {'u'};
outputs = {'x'; 'phi'};

sys_ss = ss(A,B,C,D,'statename',states,'inputname',inputs,'outputname',outputs);

poles = eig(A)
```

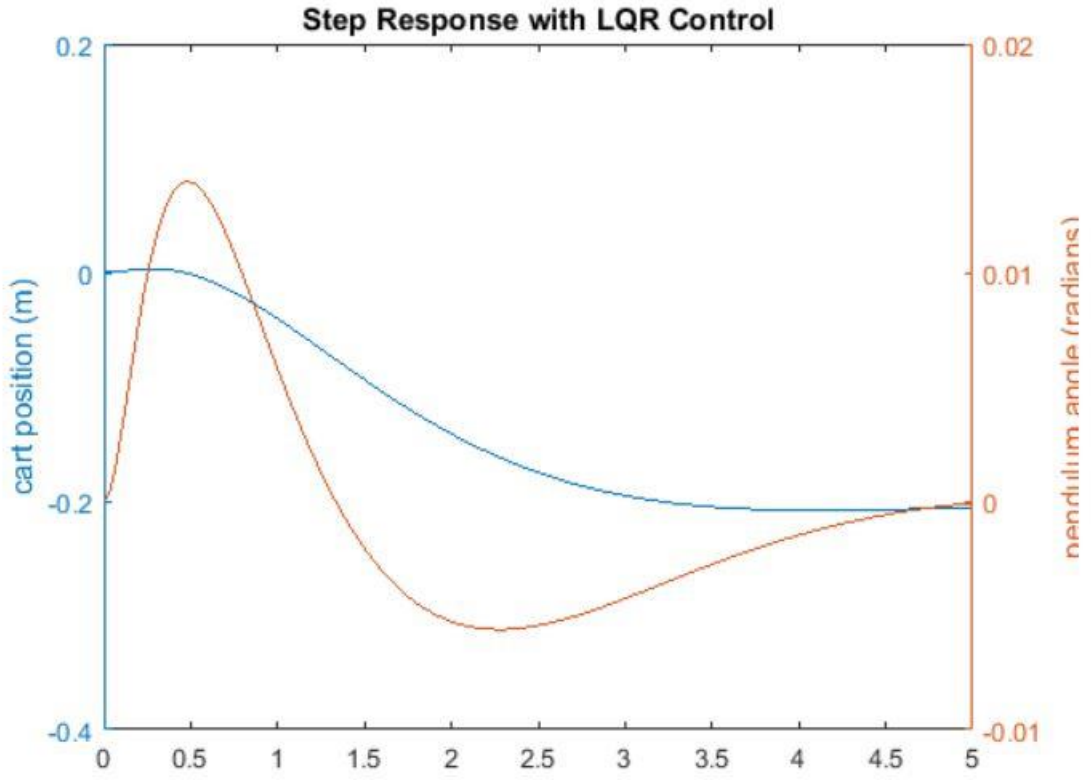
Linear Quadratic Regulation (LQR)

The next step in the design process is to find the vector of state-feedback control gains K assuming that we have access (i.e. can measure) all four of the state variables. This can be accomplished in a number of ways. If you know the desired closed-loop pole locations, you can use the MATLAB commands place or acker. Another option is to use the lqr command which returns the optimal controller gain assuming a linear plant, quadratic cost function, and reference equal to zero (consult your textbook for more details).

Before we design our controller, we will first verify that the system is **controllable**. Satisfaction of this property means that we can drive the state of the system anywhere we like in finite time (under the physical constraints of the system). For

the system to be completely state controllable, the controllability matrix must have rank n where the rank of a matrix is the number of linearly independent rows (or columns). The controllability matrix of the system takes the form shown below. The number n corresponds to the number of state variables of the system. Adding additional terms to the controllability matrix with higher powers of the matrix A will not increase the rank of the controllability matrix since these additional terms will just be linear combinations of the earlier terms.

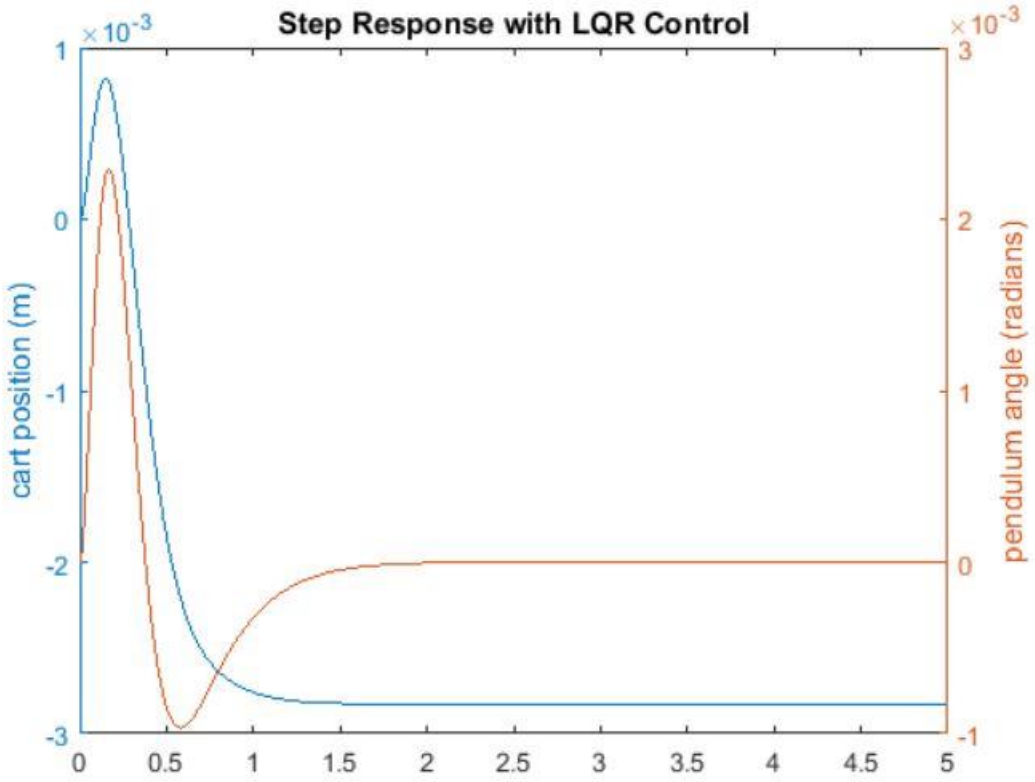
```
Q = C'*C;

R = 1;

K = lqr(A,B,Q,R)



Ac = [(A-B*K)];

Bc = [B];

Cc = [C];

Dc = [D];



states = {'x' 'x_dot' 'phi' 'phi_dot'};
inputs = {'r'};
outputs = {'x'; 'phi'};

sys_cl = ss(Ac,Bc,Cc,Dc,'statename',states,'inputname',inputs,'outputname',outputs);

t = 0:0.01:5;
r =0.2*ones(size(t));
[y,t,x]=lsim(sys_cl,r,t);
[AX,H1,H2] = plotyy(t,y(:,1),t,y(:,2),'plot');
set(get(AX(1),'Ylabel'),'String','cart position (m)')
set(get(AX(2),'Ylabel'),'String','pendulum angle (radians)')
title('Step Response with LQR Control')
```



The curve in red represents the pendulum's angle in radians, and the curve in blue represents the cart's position in meters. As you can see, this plot is not satisfactory. The pendulum and cart's overshoot appear fine, but their settling times need improvement and the cart's rise time needs to be reduced. As I'm sure you have noticed, the cart's final position is also not near the desired location but has in fact moved in the opposite direction. This error will be dealt with in the next section and right now we will focus on the settling and rise times. Go back to your m-file and change the Q matrix to see if you can get a better response. You will find that increasing the (1,1) and (3,3) elements makes the settling and rise times go down, and lowers the angle the pendulum moves. In other words, you are putting more weight on the errors at the cost of increased control effort u. Modifying your m-file so that the (1,1) element of Q is 5000 and the (3,3) element is 100, will produce the following value of K and the step response shown below.

```
Q = C'*C;

Q(1,1) = 5000;
```

```
Q(3,3) = 100

R = 1;

K = lqr(A,B,Q,R)



Ac = [(A-B*K)];

Bc = [B];

Cc = [C];

Dc = [D];



states = {'x' 'x_dot' 'phi' 'phi_dot'};
inputs = {'r'};
outputs = {'x'; 'phi'};

sys_cl = ss(Ac,Bc,Cc,Dc,'statename',states,'inputname',inputs,'outputname',outputs);

t = 0:0.01:5;
r =0.2*ones(size(t));
[y,t,x]=lsim(sys_cl,r,t);
[AX,H1,H2] = plotyy(t,y(:,1),t,y(:,2),'plot');
set(get(AX(1),'Ylabel'),'String','cart position (m)')
set(get(AX(2),'Ylabel'),'String','pendulum angle (radians)')
title('Step Response with LQR Control')
```



You may have noted that if you increased the values of the elements of Q even higher, you could improve the response even more. The reason this weighting was chosen was because it just satisfies the transient design requirements. Increasing the magnitude of Q more would make the tracking error smaller, but would require greater control force u. More control effort generally corresponds to greater cost (more energy, larger actuator, etc.).

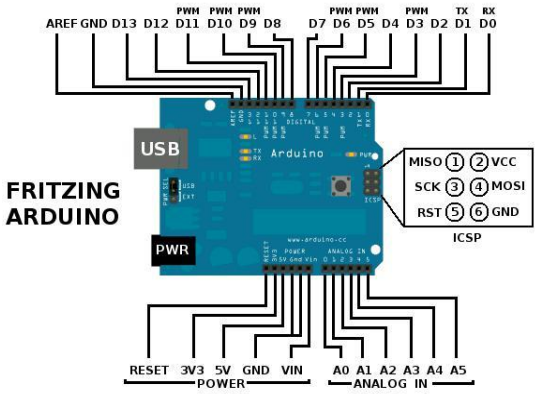
6. IMPLIMENTATION

The component used to build a working model

1. **Arduino Uno**
2. **Dc gear motor and wheel**
3. **Motor shield**
4. **MPU 6050**
5. **Battery**

Arduino Uno

Based microcontroller board based on the AT mega microcontroller 2560. It contains 8 analog inputs, 4 UARTs (hardware serial ports), 54 digital input/output pins (of which 14 can be used as PWM outputs), a 16 MHz crystal oscillator, a power jack, an ICSP header, a USB connection, as well as a reset button. It contains all on chip peripherals to support the microcontroller. By connecting it simply to a computer with a USB cable or with an AC-to-DC adapter or battery (for power purpose), to get started. Most shields designed for the Arduino Duemilanove or Decimal is compatible with Arduino. The Arduino Mega can be programmed with the 0Arduino software which is a open source software. Arduino is so vamoosed due to its open and long library. The ATmega2560 on the Arduino Mega comes pre-burned with a boot loader which helps to upload new code to it without the using an external hardware programmer. It communicates using the STK500 protocol which is also used by the AT mega



Dc Gear motor

Here I am using the dc Gear Motor with shaft Encoder, which is a high performance DC gear motor and contains magnetic Hall Effect quadrature shaft encoder. Shaft Encoder gives 90 pulses per revolution of the output shaft per channel. Motor can run from 4V to 6V supply. For shaft encoder signals,

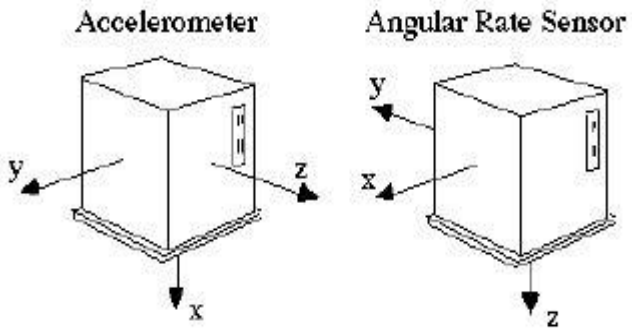
Specifications

• Voltage: 4- 6 V DC
• No Load Speed: 90 +/- 10rpm
• No Load Current: 190mA (max.250mA)
• Minimum Torque: 800 gm.cm

MPU6050

The IMU is an electronics module consist of more than one module in a single unit, which takes Angular velocity and linear acceleration data as a input and sent to the main processor. The IMU Sensor actually contains three separate sensors. The first one is the accelerometer. To describe the acceleration about three axes it generates three analog signals and acting on the planes and vehicle. Because of the physical limitations and thruster system, the significant output sensed of these accelerations is for gravity. The second sensor is the gyroscope. It also gives three analog signals. These signals describe the vehicle angular velocities about each of the sensor axes. It not necessary to place IMU at the vehicle center of mass, because the angular rate is not affected by linear or angular accelerations. The data from these sensors is collected by the microprocessor attached to the IMU sensor through a 12 bit ADC board. The sensor information communicates via a RS422 serial communications (UART) interface at a rate of about 10 Hz. The accelerometer and gyroscope within the IMU are mounted such that coordinate axes of their sensor are not aligned with self-balancing bot. This is the real fact that the two sensors in the IMU are mounted

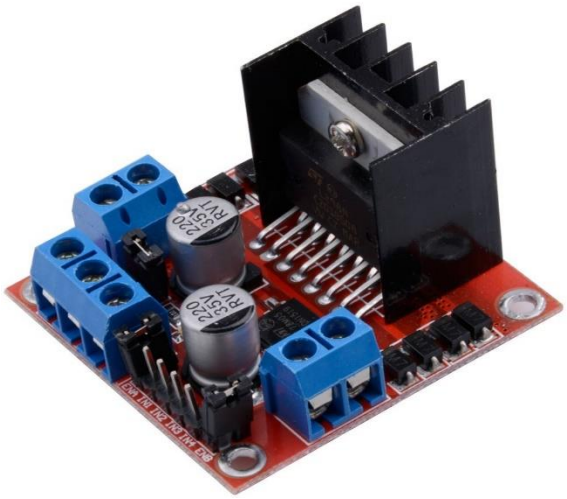
in two different orientations according to its orientation of the axis needed. The accelerometer is manufactured by using a left handed coordinate system. The transformation algorithm first uses to align the coordinate axes of the two sensors. Notice that the gyroscope are now aligned and right handed according to the IMU axis. Once the accelerometer and gyroscope axes are aligned with the axes of the IMU, then it should be aligned to vehicle reference frame



MOTOR SHIELD

Arduino is a great starting point for electronics, and with a motor shield it can also be a nice tidy platform for robotics and mechatronics. Here is a design for a full-featured motor shield that will be able to power many simple to medium-complexity projects This motor shield can power up to 4 dc motors, or 2 stepper motors plus 2 servo motors

1. 2 connections for RC servos connected to the Arduino's high-resolution dedicated timer - no jitter!
2. 4 H-Bridges: L293D chipset provides 0.6A per bridge (1.2A peak) with thermal shutdown protection, internal kickback protection diodes. Can run motors on 4.5VDC to 25VDC.
3. Up to 4 bi-directional DC motors with individual 8-bit speed selection
4. Up to 2 stepper motors (unipolar or bipolar) with single coil, double coil or interleaved stepping.
5. Pull down resistors keep motors disabled during power-up
6. Big terminal block connectors to easily hook up wires (18-26AWG) and power



7. 2-pin terminal block and jumper to connect external power, for separate logic/motor supplies
8. Tested compatible with Arduino UNO

7. Real Time implementation

Algorithm to control SBB vertically upward

a. Take the necessary header files and global variables to support the main Software.
b. Configure PID angle and speed of the motor as well as PID constants.
c. Take the read values of the IMU sensor and store it in a matrix up to 100 Samples.
d. Configure the start button, stop button and device configure button.
e. Configure PID module to update it.
f. Set digital pins, serial pins and PWM pins on the Arduino board
g. Set the kalman filter and update its weight matrix and coefficient.

1) When debug:

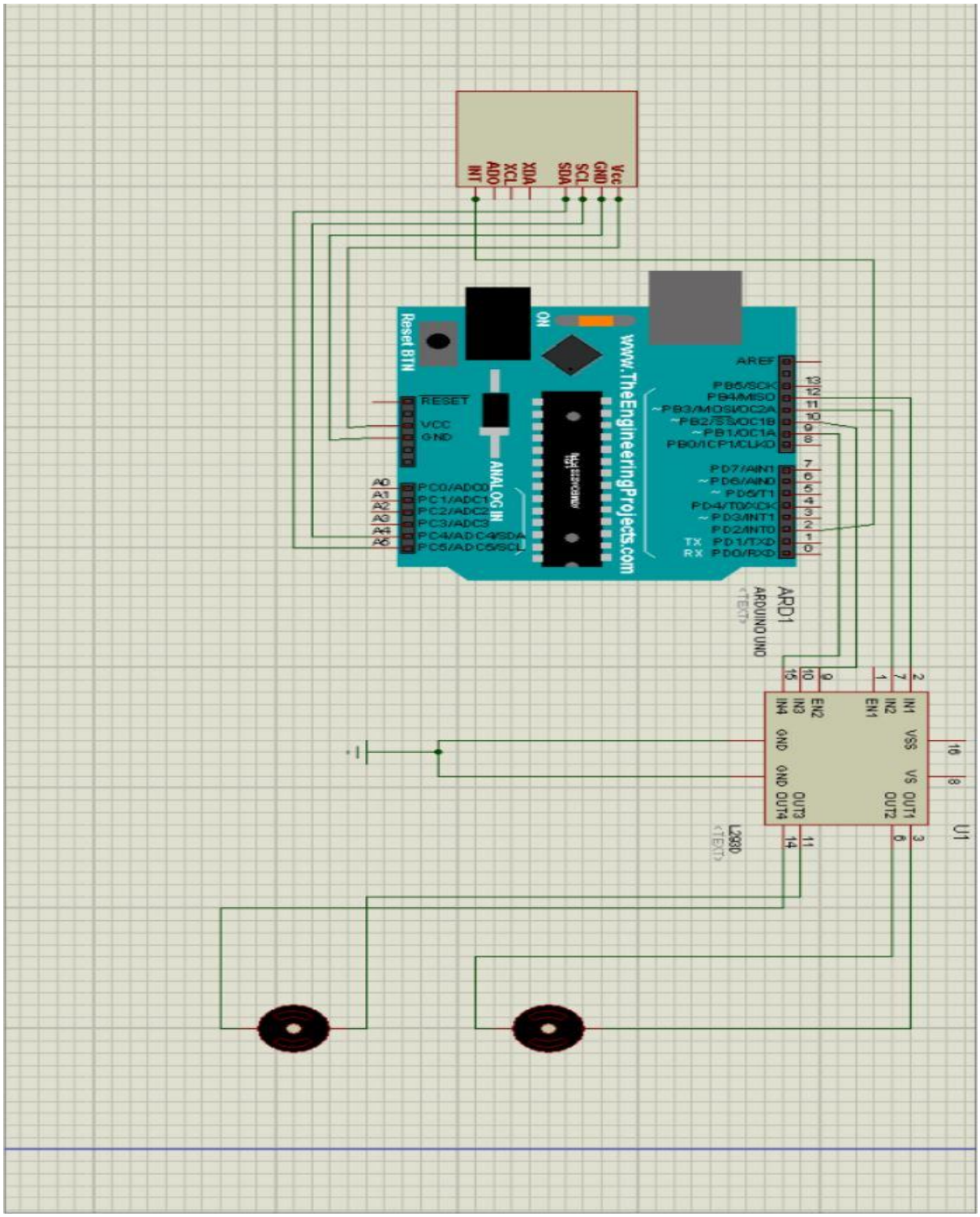
- Calibrate the previous values and update the previous declared variables.

2) When started:

1. Calculate the PID angle and motor speed.
2. According to the angel check motor speed regularly and update the IMU sensor.
3. If steering is not applied then set the external offset to zero
4. Else refer to SBB console program.

3) When stopped: do the power off. Fall automatically.

8. Schematic of electrical circuit (Proteus)



9. Budget

Name	Number	COST
Dc Gear Motor with wheel	2	70 LE
Arduino Uno	1	140 LE
MPU6050	1	70 LE
Motor Shield	1	60 LE
Wires	20	10 LE
Manufacturing material	-	30 LE
Battery	1	10 LE
TOTAL		380 LE