Stabalizing an Inverted Pendulum

Student: Kyle Gallagher Advisor: Robert Fersch

December 14, 2013

1 Introduction:

A pendulum is a simple system, consisting of a mass held aloft by either a string or a rod that is free to pivot about a certain point. This system will naturally over time approach an equilibrium where the center of mass is directly below the pivot point, and thus the energy of the system is at a minimum. However, if we invert the system, we are now presented with an inherently unstable system, the inverted pendulum, henceforth referred to as IP.

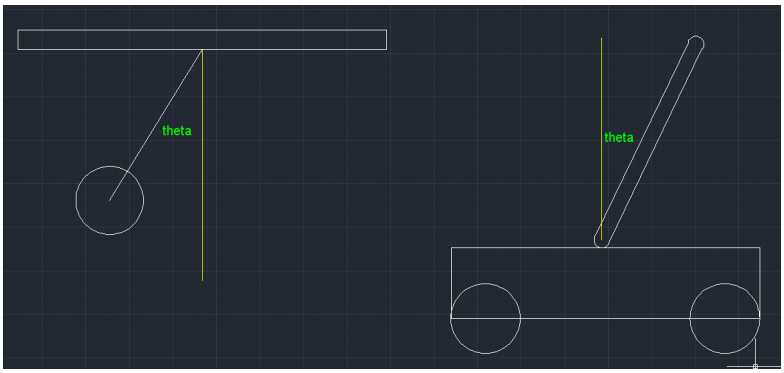


Figure 1: On the left is a diagram of a simple pendulum. On the right is a diagram of an inverted pendulum. The yellow lines indicate the normal/ reference.

While, mathematically speaking it is possible to balance a mass above a pivot point wherein the system will be at equilibrium and thus stable, this proves to be rather arduous and impractical for use beyond academic demonstrations (Figure 2). When viewed from a physics and engineering perspective, the instability of the system requires dynamic external forces for it to remain upright. Some methods for stabilizing the IP system include: applying a torque at the pivot point, displacing the pivot point along either the vertical or horizontal axis, or oscillating the mass along the pendulum. These methods work by allowing a controller to continually adjust outputs (applied forces) in reaction to changing inputs (angle of pendulum), ultimately manipulating the system towards the upright equilibrium point.

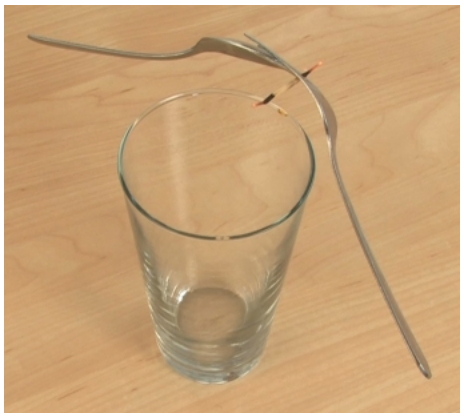


Figure 2: Two forks balencing on a toothpick. Their center of mass is directly above the point of contact between the toothpick and glass.

2 Objectives:

Similar to other capstones, this project is aimed to demonstrate critical thinking, analysis based decision making, and effective utilization of skills honed in the classroom. In addition, personal objectives included: applying principals from mathematical modeling, utilizing programing skills, gaining proficiency in professional software, and the development and fabrication of a demonstration.

3 Physics:

3.1 Simple Harmonic Oscillator

To understand the inverted pendulum, simple harmonic motion must first be understood. This system consists of a mass connected to a spring. Hooke's Law states that

$$F = kx$$

where k is the spring constant. k can be thought as a description of how stiff an associated spring may be. In the Simple Harmonic Oscillator system this force is a restoring force, therefore it may be denoted

$$F = -kx$$

this works because the equilibrium point is defined to be the origin. When the potation of a mass, x, is in the $+\hat{x}$ the force due to the spring is negative, there by pulling the mass back toward the equilibrium point. The opposite is also true.
Combining Hooke's law and Newtons laws:

$$-kx = m\ddot{x}$$

$$-m\ddot{x} - kx = 0$$

$$\ddot{x} + \frac{k}{m}x = 0$$

allowing $\omega_o^2 = \frac{k}{m}$, the equation of motion is found to be:

$$\ddot{x} + \omega_o^2 x = 0$$

This has a general solution of

$$x(t) = A\sin(\omega_o t - \gamma)$$

or

$$x(t) = A\cos(\omega_0 t - \phi)$$

where, by the trigonometric co-function identities, the phase difference between γ and ϕ is 2π.

3.2 Simple Pendulum Model

In addition to understanding simple harmonic oscillations, understanding the principles behind simple pendulums is key to understand the underlying physics behind an inverted pendulum. A simple pendulum model is similar to that of a simple harmonic oscillator, in that there exists a mass and there exists a restoring force that attempts to move said mass towards an equilibrium point (defined to be the origin). In the case of a simple pendulum there exits a mass, m, attached to a string or arm of length, l attached to a pivot point. In this case, rather than a restoring force associated with a spring, the restoring force is associated with an acceleration (gravity, g).
The resistive force is that of gravity: $F_g = mg$. As with any vector, this force may be split into its components:

$$F_{gx} = F_g\cos\theta$$

$$F_{gy} = F_g\sin\theta$$

since the mass is attached to a stationary pivot point, the force along the pendulum arm F_{gx} is canceled out by the inherent tension. Therefore the only force of note is F_{gy}. Combining this with newtons laws yields:

$$F = F_{gy}\sin\theta = -mg\sin\theta = ma \tag{1}$$

Since the position of the mass is with respect to the angle, θ, it follows that the acceleration, a, of the mass is a function of the angle. We link these together through the arc-length,S, formula:

$$S = l\theta$$

$$v = \dot{S} = l\dot{\theta}$$

$$a = \dot{v} = \ddot{S} = l\ddot{\theta}$$

Combining this value of acceleration with (1) yields:

$$-mg\sin\theta = mg\ddot{\theta}$$

$$-g\sin\theta = l\ddot{\theta}$$

rearranging,

$$l\ddot{\theta} - g\sin\theta = 0$$

$$\ddot{\theta} - \frac{g}{l}\sin\theta = 0$$

By the small angle approximation, where for $\theta < 20°$, the taylor expansion for $\sin\theta \approx \theta$, and for $\cos\theta \approx 1 - \frac{\theta^2}{2}$. Let $\omega_o^2 = \frac{g}{l}$ then the equation of motion:

$$\ddot{\theta} - \omega_o^2 \sin\theta = 0$$

whom can not be analytically solved may be expressed as

$$\ddot{\theta} - \omega_o^2\theta = 0$$

assuming the small angle approximation ($\theta < 20°$). This means that the simple pendulum model described above shows simple harmonic motion.

3.3 Inverted Pendulum Model

Rather than using Newton's law in its standard cartesian form $F = ma$, the polar form is used,

$$\tau = I\alpha = I\ddot{\theta}$$

where I is the moment of inertia around a pivot point, τ is the net torque, and α (or $\ddot{\theta}$) as the angular acceleration. Since the model consists of several components, to find the net torque the net moment of inertia needs to be found.

The moment of inertia of a rod with length, l, and mass, m_l, rotating about it's center is given by

$$I_{center} = \frac{ml^2}{12}$$

By the parallel axis theorem,

$$I_{center} = I_{point} - mr^2$$

rearranging:

$$I_{center} + mr^2 = I_{point}$$

where r is the distance between the center; the moment of inertia for the rod about the its end.:

$$\frac{m_l l^2}{12} + m(\frac{1}{2}l)^2 = \frac{m_l l^2}{12} + \frac{3m_l l^2}{12} = \frac{m_l l^2}{3}$$

$\frac{1}{2}l$ was used because that is the distance from the enter to the end (which the pendulum is allowed to rotate about).

The moment of inertia for the mass located on the pendulum, m_p, is given by:

$$I_{mp} = m_p r^2$$

where r is the distance away from the pivot point the center of given mass is located.

Recap: System knowns:

$$I_{leg} = \frac{m_l^2}{3}$$

and

$$I_{mass} = m_p l^2$$

Therefore:

$$I = I_{leg} + I_{mass} = m_c l^2 + \frac{m_l l^2}{3} = \frac{(m_p + m_l)}{3} l^2$$

Next, examine the net torque on the system. $\tau_{net} = \tau_{mass} + \tau_{leg}$

From vector calculus we know that for any two vector, $\vec{a}, \vec{b}$ the magnitude of the cross product is given by:

$$a \times b = |a||b| \sin\theta$$

Therefore:

$$\tau = r \times F = |r||F| \sin(\theta)$$

where θ is the angle between $\vec{r}$ and $\vec{f}$.

For the mass along the pendulum: $F_g = mg$ and $r = l$,

$$\tau_{mass} = r \times F \Rightarrow \tau = m_l g l \sin\theta$$

For the pendulum leg: $F = mg$ and $r = \frac{l}{2}$ (distance to center of mass),

$$\tau_{leg} = m_p \frac{l}{2} \sin\theta$$

Therefore,

$$T_{net} = m_l g l \sin\theta + m_c \frac{l}{2} \sin\theta$$

Assuming $\theta < 20°$, the small angle approximation reduces τ_{net} to:

$$\tau_{net} \approx m_l g l\theta + m_p g \frac{f}{2}\theta = (m_l g l + m_p g \frac{l}{2})\theta$$

Therefore the equation of motion $\tau = I\ddot{\theta}$:

$$(m_l g + m_p g \frac{1}{2}) l\theta = (m_c + \frac{m_l}{3}) l^2 \ddot{\theta}$$

$$(m_l g + m_p g \frac{1}{2})\theta = (m_c + \frac{m_l}{3}) l\ddot{\theta}$$

Rearranging this equation to our standard form:

$$\ddot{\theta} - \frac{(m_l g + m_p g \frac{1}{2})}{(m_c + \frac{m_l}{3}) l\ddot{\theta}}\theta = 0$$

$$\ddot{\theta} - \frac{3}{2}\frac{2m_l g + m_p g}{(3m_p + m_l)l}\theta = 0$$

$$\ddot{\theta} - \omega_0^2\theta = 0$$

where

$$\omega_0^2 = \frac{3}{2}\frac{2m_l g + m_p g}{(3m_p + m_l)l}$$

4 Methods:

4.1 Model Simplifications

The following constraints were decided upon to minimize the complexity of an IP system while retaining many of the characteristics associated with it: minimize the degrees of freedom of the system and implement a system where by the controller is remote. Minimize the degrees of freedom to one axis of travel for the cart and one axis of rotation for the pendulum was favored to decrease the complexity of the system by limiting the amount of input data the controller received. The decision to operate the cart (base of IP system) with the controller separate was made after the development of prototype 1. During the fabrication of the first prototype it became evident that fabrication time is a function of the volume of an object. By removing the controller (Arduino unit, motor shield, breadboard, power source) from the cart, the volume of said cart decreased substantially thus decreasing fabrication time while simultaneously increase the ability to manufacture prototypes as deemed necessary through the design process.

4.2 Sensors

Of the aforementioned methods to stabilize an inverted pendulum (introduction,2) each require: a reference point, the ability to discern the angle at which the pendulum may be at any given time, and the use of a controller. Since the goal of an IP system is to force the center of mass to converge to directly above it's pivot point, the normal vector or the vector perpendicular to the plane of travel (ground) is chosen to be the reference vector. The implementation of a controller necessitates an input thus mandating the use of sensors to convert physical attributes of the system into electrical signals, which the controller recognizes. Some of these sensors include gyroscopic sensors, accelerometers, light sensors, and potentiometers. Of these sensors, precision is directly proportional to the cost; with gyroscopic sensors and accelerometers being the most expensive (≈$40), followed closely by light sensors (≈ $20), and trailed by potentiometers (≈$2). Like most projects, budget constraints led to using less expensive components. This, in addition to the manner by which the controller utilizes inputs (discussed in section 4) essentially alters the opportunity cost of each sensor, by basing decisions on cost rather than accuracy and data output type.

Sensor types:

- Gyroscopic sensors: Gyroscopic sensors are able to vary output voltages that correlate to certain angles away from its normal. These sensors manage to discern slight displacements due to the underlying physical principle that a vibrating object tends to continue vibrating in the same plane regardless of its support structures orientation (4).
- Accelerometers: Accelerometers can accurately measure accelerations in up too three mutually perpendicular dimensions. They measure accelerations by calculating the displacement of a proof mass that is suspended on the surface of the integrated circuit. The output of an accelerometer is a fraction of the total voltage accepted. This voltage is proportional to the acceleration being measured (3).
- Light sensor: Light sensors emit a ray and based on the return signal are able to accurately return a voltage value, again correlated to a distance.
- Potentiometer: Potentiometers are in essence variable resistance resistors that when used as a voltage divider can return a fraction of the input voltage. Inside the potentiometer, there is a sliding contact that may move from one end of a conductive strip to the other; as the contact's position changes the measured resistance also changes. There are several variations of potentiometers including linear taper, and logarithmic taper.

4.3 Fabrication

4.3.1 Software

Due to the complex nature of the design process the decision to utilize computer aided design (CAD) programs in the design and development of desired components, was self-evident. While CAD programs are immensely powerful tools in the design process, the learning curve is steep. Those lacking the necessary training may find the design process in CAD programs incredibly arduous. Through the help of Lynda.com (a website that through video tutorials and interactive material teach individuals how to use professional software), and numerous videos on youtube.com essential skills may be acquired for little to no cost. The author was able to progress from a proverbial crawl too a jog with the use of these resources. Initially Autodesks AutoCAD was used to design two-dimensional representations of the desired components then these components were extruded, forming three-dimensional objects . This method of creating objects was satisfactory, though it left much to be desired. In AutoCAD, similar to hardcoding, if the decision to make an alteration was made, the designer would have to manipulate the two-dimensional sketches associated with each object, and re-extrude each object object. While this seems to be a minor impediment, the time required to make even the most basic changes (i.e. altering hole diameter) was substantial.

After performing several revisions and in anticipation of more, the decision was made to migrate away from AutoCAD towards Autodesks Inventor. Inventor operates on compara-

ble mechanisms of design to AutoCAD, but, treats objects in a fashion comparable to how programmers treat methods. Instead of completely redesigning an object, Inventor allows for a more fluid design process. One may reuse objects and, if necessary, completely redesign the object at its source. After a redesign, Inventor will automatically update all other occurrences of this object to reflect said changes. In addition to malleability of Inventor-based objects, the program allows designers to virtually assemble components into a final representation of their design. This is incredibly useful, as designers are able foresee complications and make appropriate adjustments on the virtual model rather completing the fabrication process then discovering issues.

4.3.2 Tools

Considering the high level of fabrication this project required, the precision of prototypes was heavily influenced by the quality and precision of tools at hand. The primary apparatus employed to fabricate custom components was the Makerbot Replicator 2, a 3D printer capable of printing with a resolution of upto 100 m. Employing the Makerbot resulted in a relatively short turn around time between design and production. The companion software for Makerbot, MakerWare, interprets STL (Standard Tessellation Language) files then prints the object by layering partially liquefied plastic. MakerWare gives the operator the ability to decide the level of infill, the number of exterior shells, layer height, temperature, travel speed, and whether or not to use rafting(see below for explanation).

- Shells: The shell, just as the name implies, comprises the exterior portion of the model. One shell consists of two (perpendicular) overlaid layers of plastic. The more shells the more freedom one has to manipulate (sand/ reshape) the final product.
- Infill: Once the shells have been placed the Makerbot will fill the interior with hexagons to ensure rigidity of the overall structure while minimizing plastic consumption.
- Rafting: strategically placed support structures used by MakerWare to reduce sagging while printing parts are not supported by previously printed layers (i.e. printing an object with a hole whose normal is parallel with the base plate)

After printing several experimental objects, it was determined to set infill to 10%, number of shells to 4, and the layer height too .25mm. The remaining variables were adjusted based on the characteristics of the filament used through experimentation.

Designs relied heavily on accurate schematics. With accurate component dimensions, high strain fabricated parts which housed sourced components could be designed to reflect expected stresses while maintaining a slender profile. The fabricated components were comprised of a potentiometer housing, a pendulum holder, and a cart. The sourced components included motors and brackets, ball casters, potentiometer, and the pendulum.

In the design of prototype 1, the schematics provided by the manufactures were heavily utilized; unfortunately, these schematics proved to utterly deplorable. Not until after the

fabricated components of prototype 1 were completely printed were they fitted. At this point the inadequate nature of the schematics was discovered. Rather than fitting together seamlessly, several of the sourced components were incompatible with the fabricated components. To mitigate these design flaws a Dremel rotary tool and miscellaneous files were used to adapt already fabricated components for use with the desired sourced components. Additional errors were generated by: prior warping of the baseplate, and design imperfections that stemmed from lack of previous design experience. These defects included lack of fudge room for fitting fabricated components together, improperly sized holes, and warping of larger components.

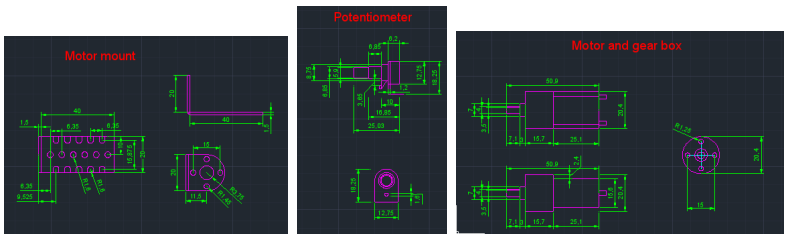


Figure 3: Component sketches prototype 2 was designed around.

In an endeavor to forgo additional discrepancies the use of Vernier calipers and alternative software was implemented. Vernier calipers, with a precision of 0.05mm, were used to establish accurate schematics of the sourced component parts. These 2D schematics (Figure 2) were constructed in AutoCAD then exported to Inventor. In Inventor the schematics were developed into 3D objects, around which prototype 2 was designed. In addition to using more appropriate design software, alterations were made to the overall design including: built-in fudge room, resized holes (125% required), and incorporation of a new bent cart frame design. The need for fudge room and larger holes now seem trivial. The bent frame however was chosen to mitigate the warping caused by the imperfect baseplate; by choosing to bend the frame the rafting, that supports the elevated sections of the cart, was able to absorb the imperfections of the baseplate.

4.3.3 Controller

A PID controller refers to a device with the capability to acquire data, interpret that data (with regards to a predefined reference), and vary output data. PIDs are useful due to the complex varied nature of real life. For example, if one were in possession of a robot that would move along a horizontal axis in one direction, and one desired for that robot to move from one position (x)(put in a diagram here) to another position (x+ dx) then the only commands necessary would be a velocity and a time duration. Unfortunately, reality is expressly more complicated. If, in the above example, the surface on which the robot traveled had differing levels of friction, a velocity and time duration would ensure its accurate arrival at the desired destination. PID controllers solve this problem by incorporating a feedback loop

thereby enabling an error to be calculated. This error is stored in memory and in conjunction with current inputs determines an appropriate output. In terms of the robot example, the destination would be defined as the reference point. After an initial command (velocity and time duration) was executed, the error, or displacement from the reference, would be calculated and a new command would be issued. PIDs are by their design, recursive, so this process will continue to iterate until no error is calculated (robot reached destination). Similar to the robot example, in the IP model developed the PID controller was the Arduino unit.

The brain of the project, the Arduino unit (Figure 3), accepted input data, truncates useless data, and then provides an output to the motors. A potentiometer was used as the input device due to its cost and since the PID controller output reflects the error or deviation from a reference; the higher levels of precision associated with alternate input sensors would have been processed in a similar method therefore the increased precision would not be necessary. The Arduino unit has both digital pins and analog pins which, in a serial connection, display voltage values from 0-255, and 0-1023 respectively. Since the Arduino unit is not able to measure the resistance of a system in and of itself, the implementation of a voltage divider is necessary (insert the math for a voltage divider here and a pic). In addition to utilizing a voltage divider, a serial connection between the Arduino unit and a computer was mandated throughout the development process, in order to visualize data in real time. The code witch enabled the Arduino unit to act as a PID used the normal vector as the reference point and with data from the potentiometer calculated the error, or angle away from the normal, the pendulum was. Using this error, the Arduino was able to issue commands dictating the speed and direction for the motors charged with displacing the base of the pendulum.

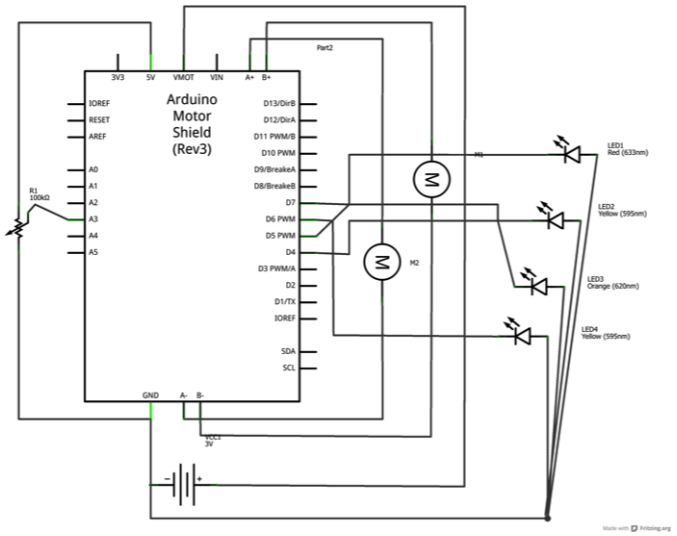


Figure 4: Pictured is a basic outline of how the controller was connected to the model.

5 Results:

One of the main objectives of this project was to design and fabricate an apparatus, which could be used for in class demonstrations. Unfortunately towards the end of the semester the heart of the model, the potentiometer, was either damaged or the interior was contaminated. Instead of providing an output voltage that could be linearly correlated to an angle, there were three distinct angles at which anomalous readings were observed. All other angles, exhibited values which were consistent with the expected readings. Two of the three anomalies occurred between 15-20 degrees away from the normal, and the third occurred directly at the normal. At these points the serial connection would display 1023 or 255 (analog inputs return 0-1023, digital returns 0-255) both indicating a lack of resistance. The PID used to control the output to the motors (insert a pic here with the data talk about what you did to mitigate the problem) interpreted these outliers as standard data and would issue appropriate commands to the motors. Due to the nature of these outliers the commands issued would be destructive to the oscillating nature of the pendulum resulting in a lack of convergence to the normal. In an effort to mitigate the effects of these outlying data points, code that truncated outliers was used in conjunction with running averages that attempted to smooth data the PID code would analyze.

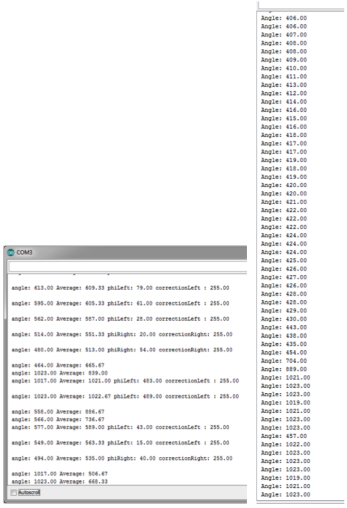


Figure 5: Examples of the serial output. This was used to attempt to mitigate outliying input values.

```
if (average == 0){
total = total - readings[index];
readings[index] = angle;
total = total + readings[index];
index = index + 1;
}
if (average > 0){
 if ( angle > 1000){
 //ignore it
 }
 else{ //angle < 1000 or = 1000
total = total - readings[index];
readings[index] = angle;
total = total + readings[index];
index = index + 1;

 }
}
if (index >= numReadings){
 index = 0;
}
average = total / numReadings;
Serial.print(" Average: ");
Serial.print(average);
```

Figure 6: Code to mitigate outlying inputs

Through out this project I learned an incredible amount. Never have I worked on a project as much as I worked on this capstone. I learned how to develop in both 2D and 3D environments, program Arduino units, use 3D printers, write in latex, and many more less notable skills. I was able to implement what I learned in physics, math, computer science, and engineering to design, and fabricate this demonstration. If I were to redo this project, I would make sure to complete the weekly journals, incorporate a gyroscopic sensor into the pendulum to better determine the normal, and stretch the project over two semesters instead of one.

6 Bibliography

1. Classical Dynamics of Particles and Systems 5th edition; Thornton and Marion
2. Wikipedia: "Inverted pendulum":
   http://en.wikipedia.org/wiki/Inverted_pendulum#
   Examples_of_inverted_pendulums
3. SensorWiki.com, "accelerometer":
   http://sensorwiki.org/doku.php/sensors/accelerometer
4. Gyroscopes:
   http://en.wikipedia.org/wiki/MEMS_gyroscope#MEMS_gyroscope

5. EasyCAD for you:
   http://www.youtube.com/watch?v=xquI8gcdwbs

6. Lynda.com

7. Numerical Solutions to Differential Equations:
   http://introcs.cs.princeton.edu/java/94diffeq/

8. Classical Dynamics: Taylor

7 Additional Diagrams:

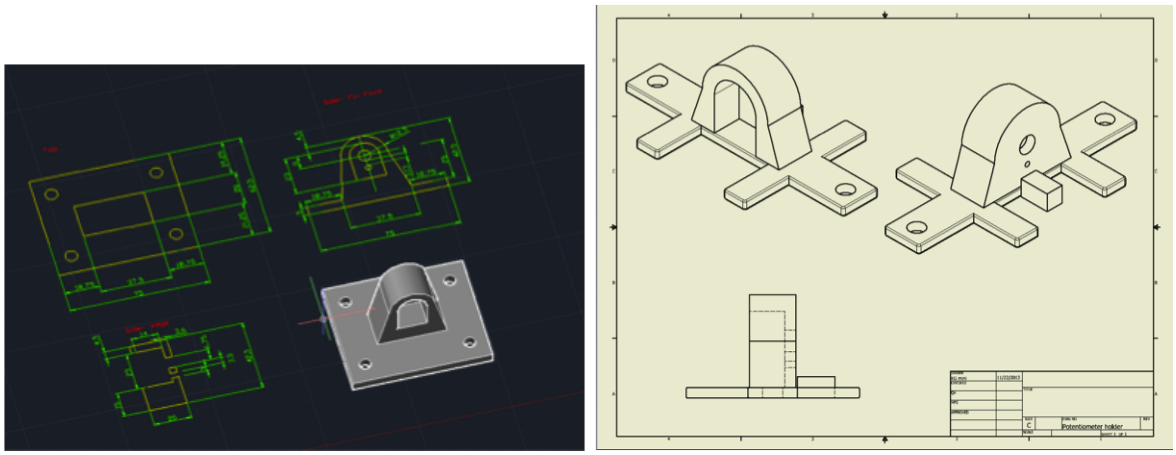


Figure 7: Top: prototype 1 potentiometer holder. Bottom: prototype 2 potentiometer holder

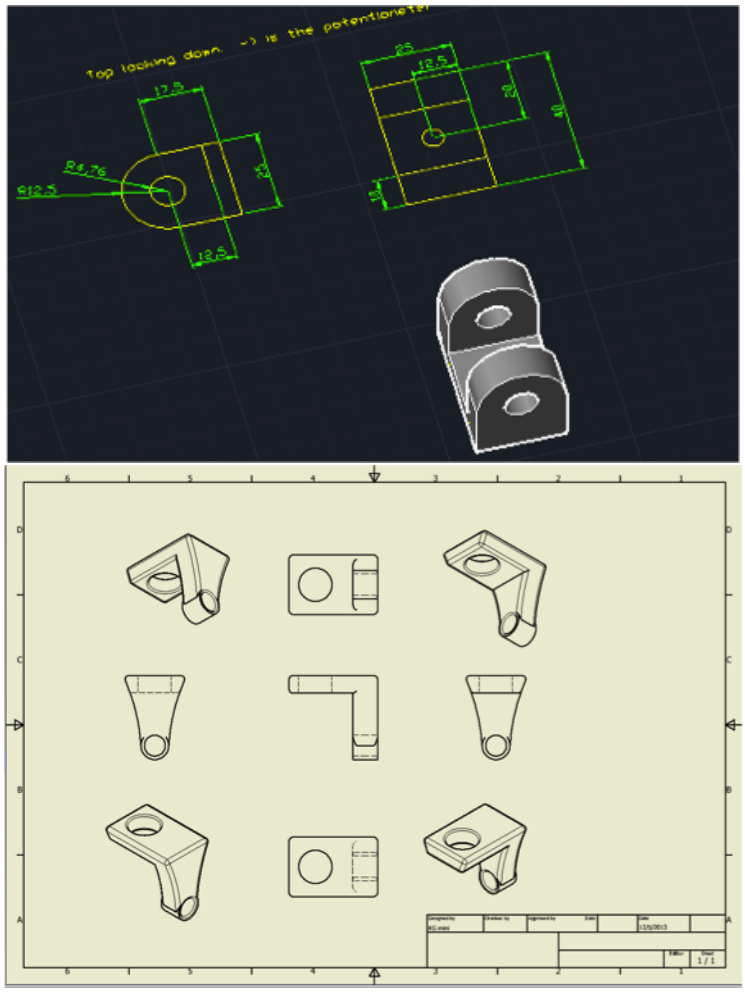
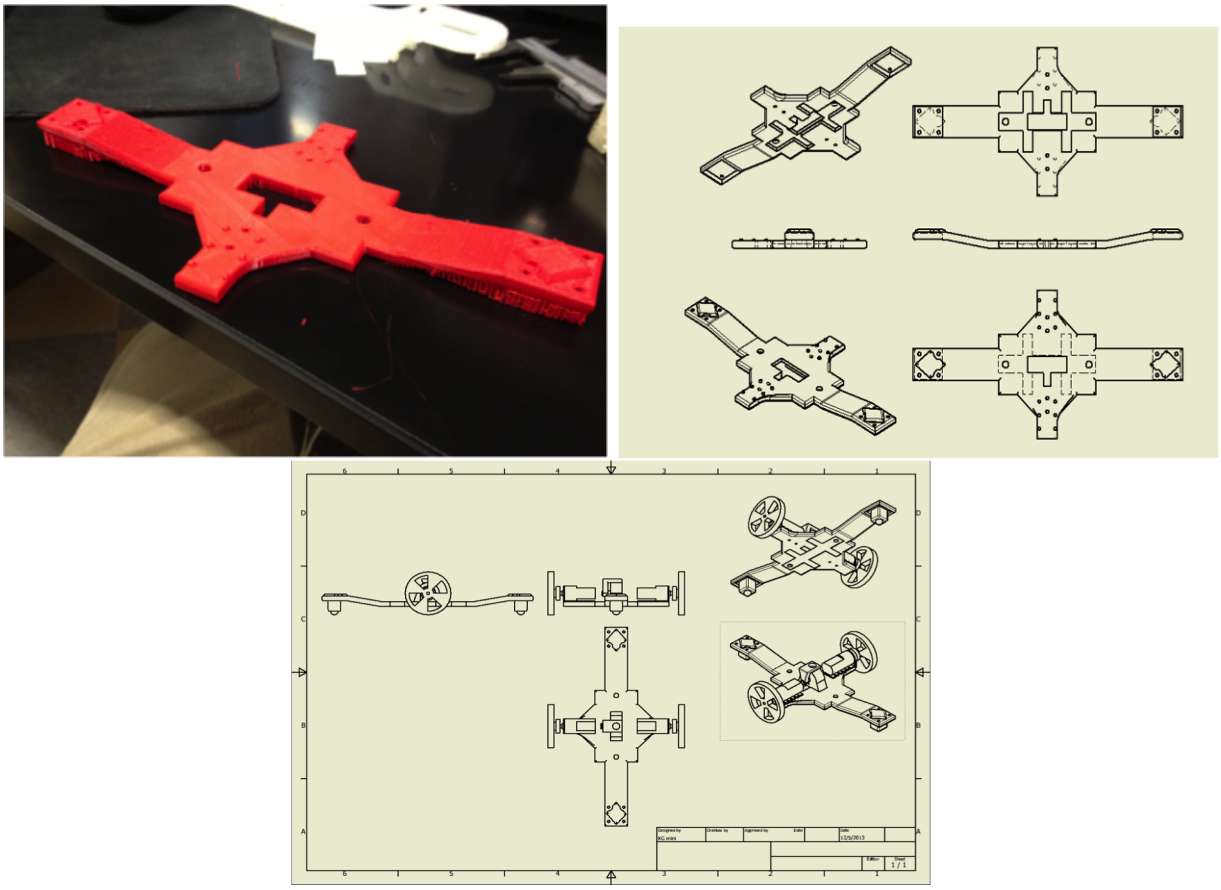


Figure 8: Top: prototype 1 pendulum holder. Bottom: prototype 2 pendulum holder

Figure 9: Top: prototype 2 with rafting still present (rough facing down). Middle: diagram of cart. Bottom: assembly of cart including components



```
//Code: Inverted Pendulum

//int digitalSensor = 2;
int AsensorPin = A3;       //conect potentiometer to A3
//int ledPinRed = 4; // connect led to D4
//int ledPinYellowLeft = 5;
//int ledPinGreen = 6;
//int ledPinYellowRight = 7;

const int mADir = 12;
const int mASpeed = 3;
const int mABrake = 9;

const int mBDir = 13;
const int mBSpeed = 11;
const int mBBrake = 8;

//float lVolt =0;
//float hVolt = 5;

float normal =534; //(hVolt-lVolt)/2;

float errorLeft = 0;
float errorRight = 0;

//smoothing setup
const int numReadings = 3;
float readings[numReadings];
int index = 0;
float total = 0;
float average = 0;
int fudge = 50;

//PID info
float PID = 0;
int input = 0;
int lastInput = 0;
float kP = 20;//57
float kI = 10;//26
float kD = 5;//12

void setup(){
  Serial.begin(9600);
  //pinMode(ledPinRed, OUTPUT);
  //pinMode(ledPinYellowLeft, OUTPUT);
```

```
  //pinMode(ledPinGreen, OUTPUT);
  //pinMode(ledPinYellowRight, OUTPUT);

  //Setup motor A
  pinMode(mADir, OUTPUT);
  pinMode(mASpeed, OUTPUT);
  pinMode(mABrake, OUTPUT);

  //setup motor B
   pinMode(mBDir, OUTPUT);
  pinMode(mBSpeed, OUTPUT);
  pinMode(mBBrake, OUTPUT);

  for( int thisReading = 0; thisReading < numReadings; thisReading++){
  readings[thisReading] = 0;
  }

  //setup pin 0 for reading input
  pinMode(digitalSensor, INPUT);
}


void loop(){
  //boolean dir;
  //float motorSpeed ;
  float voltage;
/*
  //removed because analog has better resolution
  float v2 = getVoltage(digitalSensor);
   //v2 = mapFloat(v2, 0,5,0,1024);
   Serial.print("Digital voltage: ");
   Serial.print(v2);
   */

/*
  voltage = getVoltage(AsensorPin);
  Serial.print(" Analog voltage: ");
  Serial.print(voltage);
*/

 /* //Blink the RED LED based on voltage
  char buffer[5];
  int sensorValue = analogRead(AsensorPin);
  Serial.print(" sensorValue: ");
  Serial.print(sensorValue);
  digitalWrite(ledPinRed, HIGH);
  delay(voltage);
```

```
  digitalWrite(ledPinRed, LOW);
  delay(voltage);
*/


  float angle = analogRead(AsensorPin);
  //double angleD = angle;
  //float angle = mapFloat(voltage,lVolt,hVolt,0,255); //map the voltage (0-5) to the digital
out (0-255) for the motors (1.16/4.23 because of the potentiometer holder's angle
constraints experementation values *can build in min and max feture)
  //angle = constrain (angle, 0,255);

  Serial.print(" angle: ");
  Serial.print(angle);

  if (average == 0){
  total = total - readings[index];
  readings[index] = angle;
  total = total + readings[index];
  index = index + 1;
  }
  if (average > 0){
   if ( angle > 1000){
   //ignore it
   }
   else{ //angle < 1000 or = 1000
  total = total - readings[index];
  readings[index] = angle;
  total = total + readings[index];
  index = index + 1;

   }
  }
  if (index >= numReadings){
   index = 0;
  }
  average = total / numReadings;
  Serial.print(" Average: ");
  Serial.print(average);

  //use average to get rid of outliers
  if ((angle > (average-fudge)) && (angle < (average + fudge))){

  //pendulum too far to the left
  if (angle > (normal)){
    //digitalWrite(ledPinYellowLeft, HIGH);
    //digitalWrite(ledPinYellowRight, LOW);
```

```
    //digitalWrite(ledPinGreen, LOW);

    float offLeft = getPhi(angle);
    Serial.print(" phiLeft: ");
    Serial.print(offLeft);
    float correctionLeft = calcPid(offLeft);
    Serial.print(" correctionLeft : ");
    Serial.println(correctionLeft);

    analogWrite(mASpeed, correctionLeft);
    analogWrite(mBSpeed, correctionLeft);
    digitalWrite(mADir, LOW);
    digitalWrite(mBDir, LOW);
    digitalWrite(mABrake, LOW);
    digitalWrite(mBBrake, LOW);

    //dir = true;           //true == HIGH
    //motorSpeed = correctionLeft;

   }
  /*  if (angle == normal){
    digitalWrite(ledPinGreen, HIGH);
    digitalWrite(ledPinYellowLeft, LOW);
    digitalWrite(ledPinYellowRight, LOW);

    }*/
    if(angle < (normal)){
     //digitalWrite(ledPinYellowRight, HIGH);
     //digitalWrite(ledPinYellowLeft, LOW);
     //digitalWrite(ledPinGreen, LOW);

      float offRight = getPhi(angle);
      Serial.print(" phiRight: ");
      Serial.print( offRight);
      float correctionRight = calcPid(offRight);
      Serial.print(" correctionRight: ");
      Serial.println(correctionRight);

      analogWrite(mASpeed, correctionRight);
      analogWrite(mBSpeed, correctionRight);
      digitalWrite(mADir, HIGH);
      digitalWrite(mBDir, HIGH);
      digitalWrite(mABrake, LOW);
      digitalWrite(mBBrake, LOW);

    }
    }
```

```
  delay(1);
Serial.println();
  return;
}



//implement the PID
  float calcPid (float input){
    PID =(kP * input) + (kI*(input + lastInput)) + (kD*(input-lastInput));
    lastInput = input;
    float output = constrain(PID,0,255);
    return output;
  }


//Phi is the distance from the normal
float getPhi(float angle){
  float phi = 0;
  if (angle > normal){
    phi = (angle - normal);
  }
  else if(angle < normal){
    phi = (normal - angle);
  }
  //phi = mapFloat(phi, 0, normal, 0, 255);
  //phi = constrain(phi, 0, 250);    //limit to 250 for symetry
  return phi;
}

float getVoltage(int pin){
  float vIn = analogRead(pin);
  return( vIn * 0.004882814);
}
//takes input and maps it in float
float mapFloat(float input, float inMin, float inMax, float outMin, float outMax){
  float y = ((outMax-outMin)*((input - inMin)/(inMax-inMin)))+outMin;
  return y;
}
```

Inverted Pendulm: Expenses

Item	QTY	$/	Net Cost	URL	Site	Paid	Recived
Wheels (60mm diameter)	2	7.95	15.9	http://www.pololu.com/catalog/product/1421	pololu	yes	y
Arduino Uno start kit	1	54.99	54.99	http://www.amazon.com/gp/product/B00BT0NDB8/ref=	amazon	yes	y
Ball Caster	2	5.99	11.98	http://www.pololu.com/catalog/product/66	pololu	yes	y
Motor wheel mount hub 4mm (2 per)	1	6.95	6.95	http://www.pololu.com/catalog/product/1081	pololu	yes	y
Metal motor Bracket (2 per)	1	6.95	6.95	http://www.pololu.com/catalog/product/1138	pololu	yes	y
Motor (73:1) 4mm shaft	2	19.95	39.9	http://www.pololu.com/catalog/product/1109	pololu	yes	y
#4-40 screws Phillips (25 pack)	1	0.89	0.89	http://www.pololu.com/catalog/product/1962	pololu	yes	y
Filament 1.75mm ABS (2.2 Pounds)	1	30	30	http://www.amazon.com/Signstek-Filament-Printers-Ma	amazon	yes	y
Potentiometer (10K Ohm)	3	0.95	2.85	https://www.sparkfun.com/products/9939	Sparkfun	yes	y
Motor Driver shield	1	25.01	25.01	http://www.amazon.com/Arduino-Motor-Shield-R3/dp/B	amazon	yes	y
pendulum	1	8.99	8.99	ace hardware		na	na
Shipping			32.19				
	Total Cost		236.6				

Order / **Shipping**	1	2	3
pololu	15	4.95	4.95
sparkfun	3.7	3.69	0
Amazon	0		
total	19	8.64	4.95
		sum	32.19