CAT FLIPPING

ARDEN, SEAN, AND WILL

1. Moment of Inertia of the Cat

We model a falling cat with two cylinders A and B connected by a spherical joint (which we assume to be massless) with A and B separated by angle θ.

Claim 1. About the center of mass, the y axis of the cat is principle. Further, we can write $\vec{L} = (2I_{yy})\omega_y \hat{y}$. For an I_{yy} dependent on the geometry of the cylinders and θ.

Proof. Let the moment of inertia tensors of cylinders A and B about the center of mass of the cat as a whole be denoted I_A and I_B. Recall the definition of the moment of inertia tensor:

$$I = \begin{pmatrix} \sum(y^2+z^2) & \sum xy & \sum xz \\ \sum yx & \sum(x^2+z^2) & \sum yz \\ \sum zx & \sum zy & \sum(x^2+y^2) \end{pmatrix}$$

Then, by the reflectional symmetry of bodies A and B, the calculations for I_A and I_B are identical except for the replacement of y by $-y$. So, we can safely write

$$I_A = \begin{pmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{pmatrix} \text{ and } I_B = \begin{pmatrix} I_{xx} & -I_{xy} & I_{xz} \\ -I_{yx} & I_{yy} & -I_{yz} \\ I_{zx} & -I_{zy} & I_{zz} \end{pmatrix},$$

And so the moment of inertia tensor I of the cat as a whole is of the form

$$I = I_A + I_B = \begin{pmatrix} 2I_{xx} & 0 & 2I_{xz} \\ 0 & 2I_{yy} & 0 \\ 2I_{zx} & 0 & 2I_{zz} \end{pmatrix}$$

Then $\vec{y}$ is a clear eigenvector of I confirming that the y axis is a principle axis. □

Note that the above argument can be generalized to any two bodies with reflectional symmetry through the x, y plane.

Claim 2. The I_{yy} in claim 1 can be written $I_{y'y'}\cos^2(\theta/2) + I_{z'z'}\sin^2(\theta/2)$ where $I_{y'y'}$ and $I_{z'z'}$ are the moment of inertia elements corresponding to the principle axes.

Proof. Let x', y', z' denote the principle axes of cylinder A about the center of mass with z' going down the length of the cylinder, x' parallel to x, and y' in the corresponding location for a right-handed orthogonal coordinate system. Then, with this coordinate system corresponding to the principle axes, we can write the moment of inertia tenor I'_A of cylinder A about the center of mass of A.

$$I_A = \begin{pmatrix} I_{x'x'} & 0 & 0 \\ 0 & I_{y'y'} & 0 \\ 0 & 0 & I_{z'z'} \end{pmatrix}$$

Now, we adjust to the x, y, z coordinate system established above and solve for I_{yy}. Note $\hat{y}$ is given by $(0, \cos(\theta/2), \sin(\theta/2))$ in x', y', z' coordinates. Thus by adjusting coordinates we have

$$\begin{aligned} I_{yy} &= \begin{pmatrix} 0 & \cos(\theta/2) & \sin(\theta/2) \end{pmatrix} \begin{pmatrix} I_{x'x'} & 0 & 0 \\ 0 & I_{y'y'} & 0 \\ 0 & 0 & I_{z'z'} \end{pmatrix} \begin{pmatrix} 0 \\ \cos(\theta/2) \\ \sin(\theta/2) \end{pmatrix} \\ &= I_{y'y'} \cos^2(\theta/2) + I_{z'z'} \sin^2(\theta/2) \end{aligned}$$

The above gives the value of I_{yy} about the center of mass of A, but the center of mass of the cat as a whole shares identical x and z values. So, by $I_{yy} = \sum x^2 + z^2$, the value of I_{yy} holds about the center of mass the cat.

□

Again, note the above argument is generalizable given the principle axes.

Then overall we have a simple relationship between the angular momentum of the cat and an angular velocity ω of the cat in the $\vec{y}$ direction.

$$\vec{L} = (I_{y'y'} \cos^2(\theta/2) + I_{z'z'} \sin^2(\theta/2))\omega\hat{y} \tag{1}$$

In particular, if A and B are two solid cylinders of equal density with length L and radius R, we have

$$\vec{L} = (/ * */) \tag{2}$$

2. ????

3. Numerical Example

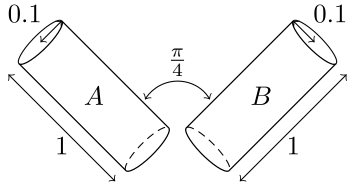

Figure 1. Specific model of cat for example

Here we will consider a physical example using our model of a cat, to compute all of the neccessary values. We will consider a cat of `mass` $= m = 1$, `length` $= l = 1$, radius `radius` $= r = 0.1$, and maximum angle of bend $\theta = \frac{\pi}{4}$. We will also consider the cat to be droped from a height of 10, and with standard gravity of -9.8. The model of this cat is depicted in Figure 1.

The first step is to compute the time that the cat has to do the rotation. We find this simply using basic kinematics

$$\Delta t = \sqrt{\frac{2h}{g}} = \sqrt{\frac{2 \cdot 10}{9.8}} \approx 1.428.$$

Next we will also compute the moment of inertia I, and to compute this we require the inertia tensor for each of the cylinders individually. So we compute these to be

$$\begin{pmatrix} \frac{m}{12}\left(r^2+l^2\right) & 0 & 0 \\ 0 & \frac{m}{12}\left(r^2+l^2\right) & 0 \\ 0 & 0 & \frac{mr^2}{2} \end{pmatrix} \approx \begin{pmatrix} 0.084 & 0 & 0 \\ 0 & 0.084 & 0 \\ 0 & 0 & 0.005 \end{pmatrix}.$$

Since both A and B cylinders are identical, then this is the interia tensor for both of them. The next stage is to compute the moment of inertia in the $\vec{y}$ axis.

$$I_{yy} = I_{y'y'} \cos^2\left(\frac{\theta}{2}\right) + I_{z'z'} \sin^2\left(\frac{\theta}{2}\right) \approx 0.071841 + 0.000732 \approx 0.0726.$$

Then from our previous computation, we evaluate for the rotational speed of the cylinders ω. We compute this as

$$\omega = \frac{\pi}{\Delta t \left(1 - \frac{2I_{z'z'}}{2I_{yy}} \sin\left(\frac{\theta}{2}\right)\right)} \approx 2.2514.$$

This is the rate that each cylinder must rotate at inorder for the full rotation to occure.

To verify that this rate will work, we compute /*TODO*/

4. Simulation

To verify our computations, we constructed a software simulation, that would be easily configurable, to allow rapid development. For the simulation we chose to use the `Python` programming language, and to make use of the `pyBullet` physics simulator. `pyBullet` is provided using python's integrated package manager `pip`.

4.1. **URDF File Format.**

`pyBullet` makes use of the `URDF` file format, so all models in the simulation must be loaded from a urdf file. This format has some specific constraints that are explained here, and the cat model that we used is included. The urdf file type is a subset of xml.

The model is constructed with `links` and `joints`, and the entier format is contained within a `robot` object.

4.1.1. *Link.*

The link object can contains three sub objects, `visual collision`, and `intertial`. The visual and collision objects, can define a `geometry`, which can in turn be constructed from a few selected primatives. And they can also define an `origin` object. The geometry is used to determin the visual display, or the collision geometry accordingly, and the origin object defines the orientation and the position of the link relative to the robots origin.

The `inertial` object can define the componets mass, and its inertial tensor, along with the origin for the inertial computation. For most purposes the origins for all three of these components can be the same, and the geometry of the visual and collision components can also be the same.

4.1.2. *Joint.* The joint object defines the connections between the different links. Each joint has a parent link and a child link, and an origin. The types of joints that are available are a fixed joint, where no motion is permitted, a revolute joint, which is rotation about a single axis, and a slide joint, which is an in and out motion. Each of these different types of joints have different parameters that need to be defined for each of them. Our model only makes use of the fixed joints, and the revolute joints.

For the revolute joint, we must also define the axis for its rotation and the limits of angles for the rotation.

4.1.3. *Construction.* To construct the cat with variable parameters, we wrote a script to generate the `urdf` file altering the parameters to be what we desired.

4.2. **Simulation Setup.** `pyBullet` provides all of the physics simulations that are necessary for most common applications, but first it must be initialized in the code. To do this one must use the following commands before any physics simulation is possible.

```
import pybullet as p
physicsClient = p.connect(p.GUI)
p.setAdditionalSearchPath(pybullet_data.getDataPath())
```

These commands initialize the graphical interface for the simulation, and includes a path for some standard `pyBullet` models.

The next step is to set global constants that are persistent for the entirety of the simulation. In this case we will only set the gravitational constant.

```
p.setGravity(0,0,-9.8)
```

This function sets the gravitational force in the x, y, and z directions.

The final step before beginning the simulation is to load any models that will be used in the simulation. For our purposes, we load the standard `plane.urdf`, and we load our constructed `cat.urdf`.

```
planeId = p.loadURDF("plane.urdf")
startPos = [0,0,10]
startOrientation = p.getQuaternionFromEuler([0,0,0])
catId = p.loadURDF("cat.urdf",startPos, startOrientation)
```

These functions load the plane model positioned at the origin, and loads the cat model positioned at $(0, 0, 10)$, with the default orientation.

4.3. **Simulation Computation.** With our our organization of the simulation, we are able to compute all neccesary values prior to the executation of the simulation. This process is implementing the equations that we found priviously, and computing their numerical values for the provided model of the cat. The first stage is computing the time that the cat has to preform the menuver, and partitioning that time for the three different stages of the motion.

```
t_max = np.sqrt(2 * 10 / 9.8)
step_2 = 2 * t_max / 10
step_3 = 8 * t_max / 10
dt = (step_3 - step_2)
```

Next we compute the moment of inertia.

```
  ix = 1/12 * mass * (radius**2 + length**2)
  iy = 1/12 * mass * (radius**2 + length**2)
  iy = 1/2 * mass * radius**2
  ic = iy * np.cos(theta / 2.0)**2 + iz * np.sin(theta / 2.0)**2
  c = 1 - 2 * iz / (2 * ic) * np.sin(theta / 2.0)
```

These first lines are computing the moment of inertia in the three principal axes for each individual cylinder of the cat model. Next we compute the combined moment of inertia, and a constant that we call c. This constant c we derived previously. Now using the moment of inertia, and the time available for the cat, we compute the rate at which the cat should rotate. Since this will be variable with respect to time, we also construct a function to return the rate of rotation.

```
  omega_0 = np.pi / (dt * cloud)
  omega = lambda t: omega_0 * (-np.cos(np.pi / dt * t) + 1)
```

4.4. **Simulation Execution.** Now with all the components of the simulation constructed, we are able to run the simulation. The entire code is presented below, and we will explane the three stages of execution afterwards.

```
while i < 1000:
  p.stepSimulation()
  if t < step_2:
    p.setJointMotorControlArray(catId, [0,2], p.POSITION_CONTROL,
    targetPositions=[(np.pi - theta) / 2.0 * (1 - np.cos(np.pi / step_2 *
    (t - step_2))),0])
  elif t < step_3:
    p.setJointMotorControlArray(catId, [0,2], p.POSITION_CONROL,
    targetPositions[(np.pi - theta) * np.cos(phi), (np.pi - theta) * np.sin(phi)])
    phi += omega(t - step_2) / fps
  elif t < t_max:
    p.setJointMotorControlArray(catId, [0,2], p.POSITION_CONROL,
    targetPositions=[x/2 * (1 + np.cos(np.pi / dt_3 * (t - step_3))), y/2 * (1
    + np.cos(np.pi / dt_3 * (t - step_3)))])
  t += 1.0 / fps
  i += 1
```

The simulation can be broken into three steps. The first is the process of bending the straight cat into the partialy bent position. Then the next stage is the rotational motion, which results in the reorientation. THen the final stage is the straightening of the cat.