



# REACTION-WHEEL STABILIZED BICYCLE ROBOT

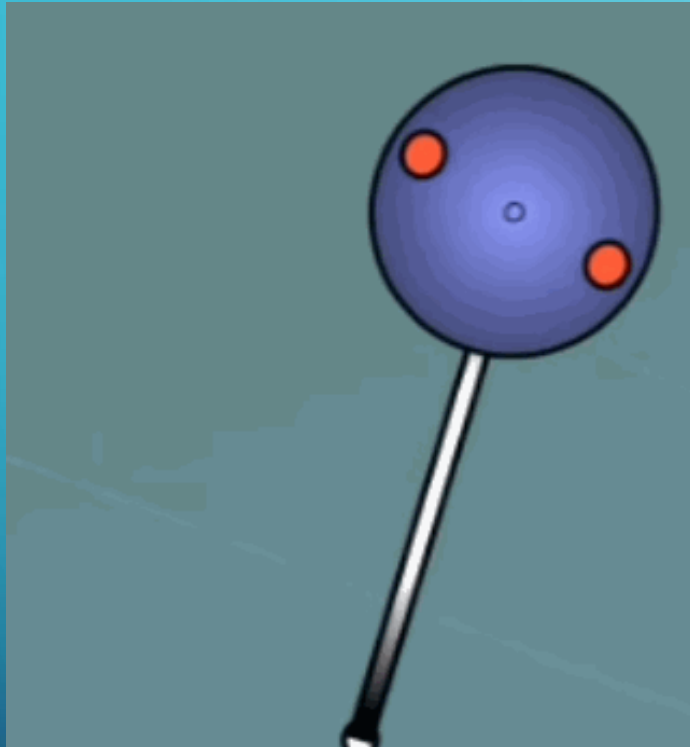
INTRODUCTION TO ROBOTICS - EGN4060C

ERIC BAKER

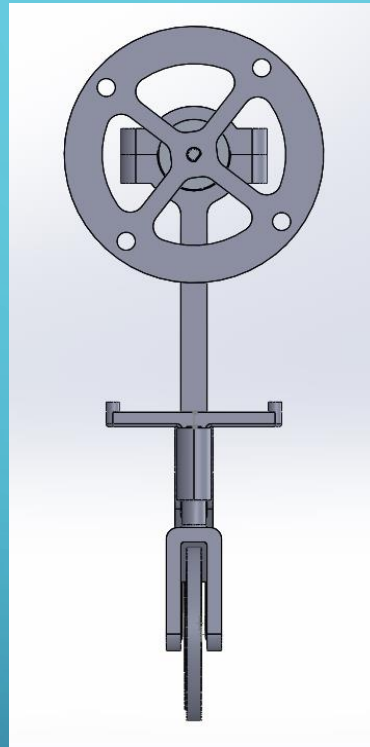
BRETT BURGESS

SALOMON HASSIDOFF

# INTRODUCTION



Reaction-wheel pendulum animation



Frontal view of  
CAD assembly

- Our project consist of a bicycle robot that stays upright without toppling over by using a reaction wheel.
- The system is modeled as an inverted pendulum system. Like most inverted pendulum systems, this project heavily focused on dynamics and controls.
- Reaction-wheels work based on conservation of angular momentum. When the systems tips over, the reaction-wheel imparts a counter-torque to rotate the system back to equilibrium.

# ROBOTS THAT MAKE USE OF REACTION-WHEELS



**Murata Boy:**

[https://www.youtube.com/watch?v=G3\\_0OzaoQ00](https://www.youtube.com/watch?v=G3_0OzaoQ00)



**Murata Girl:**

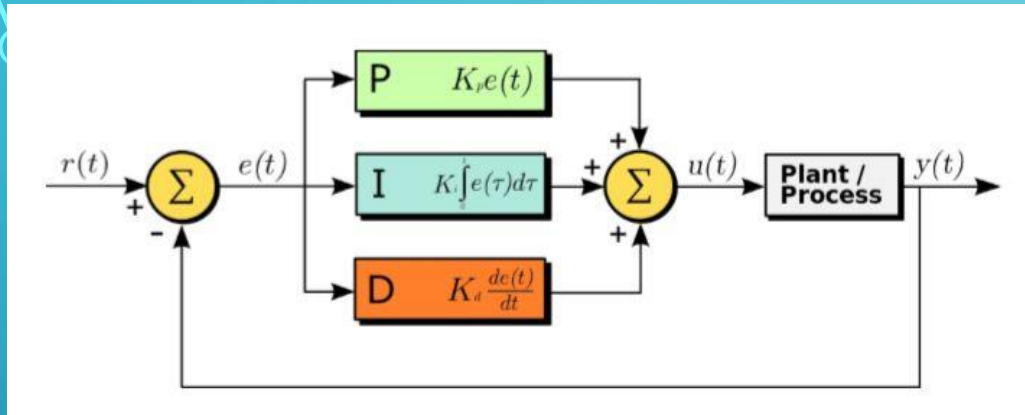
<https://www.youtube.com/watch?v=IAWYgZbUvHs&t=96s>



**Cubli:**

[https://www.youtube.com/watch?v=n\\_6p-1J551Y](https://www.youtube.com/watch?v=n_6p-1J551Y)

# SOFTWARE DESIGN



Control loop

Our algorithm utilizes a PID controller inside a feedback loop to maintain a desired tilt angle.

- $r(t)$ : desired output
- $y(t)$ : actual output
- $e(t)$ : error
- $u(t)$ : controller output / system input

```
27 // PID and controls values
28 double Kp = 3;      // Proportional gain
29 double Ki = 0.01;   // Integral gain
30 double Kd = 0.5;    // Derivative gain
31 double setpoint;    // Desired value in degrees
32 double input;       // Inclination angle from IMU
33 double output;      // Reaction wheel spin for counter torque
34 PID myPID(&input, &output, &setpoint, Kp, Ki, Kd, DIRECT); // PID instance
```

CL RESPONSE	RISE TIME	OVERSHOOT	SETTLING TIME	S-S ERROR
Kp	Decrease	Increase	Small Change	Decrease
Ki	Decrease	Increase	Increase	Eliminate
Kd	Small Change	Decrease	Decrease	No Change

PID Tuning

The PID parameters are adjusted until optimal system behavior is achieved.

**Proportional:** looks at present error to provide quick response

**Integral:** looks at past error to eliminate steady-state error

**Derivative:** predicts future error to prevent overshoot



# IMPLEMENTATION

- Gyro is prone to drift.
- Accelerometer is sensitive to vibration.
- A Kalman filter is an optimal estimation algorithm that fuses the measurements for a more accurate estimate of the position.

```
set PID parameters (kp, ki, kd)
set desired roll angle
set max motor voltage

loop
{
    function getIMUangle()
    {
        read gyro data
        apply Kalman filter

        return roll angle
    }

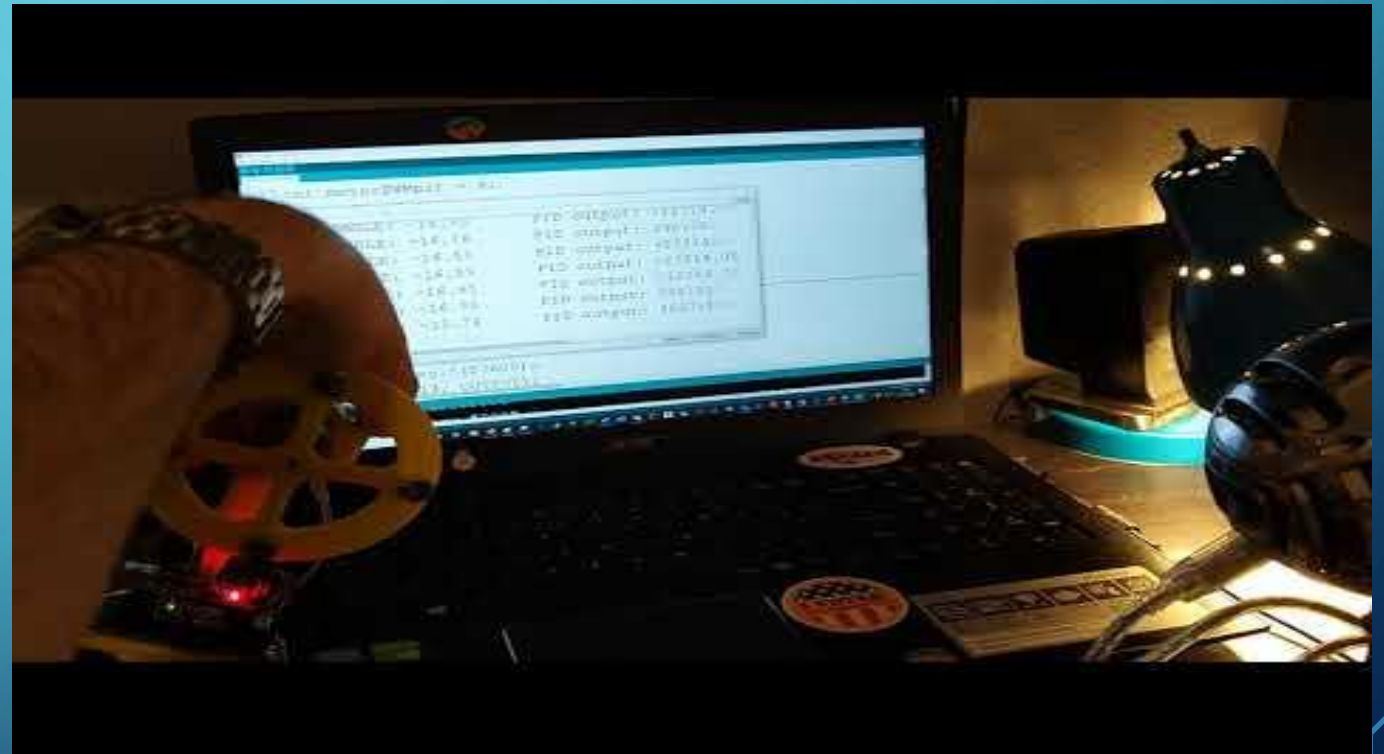
    error equals desired angle minus measured angle

    P equals kp times error
    I equals ki times integral of error
    D equals kd times derivative of error

    controller output equals sum of P, I, D terms

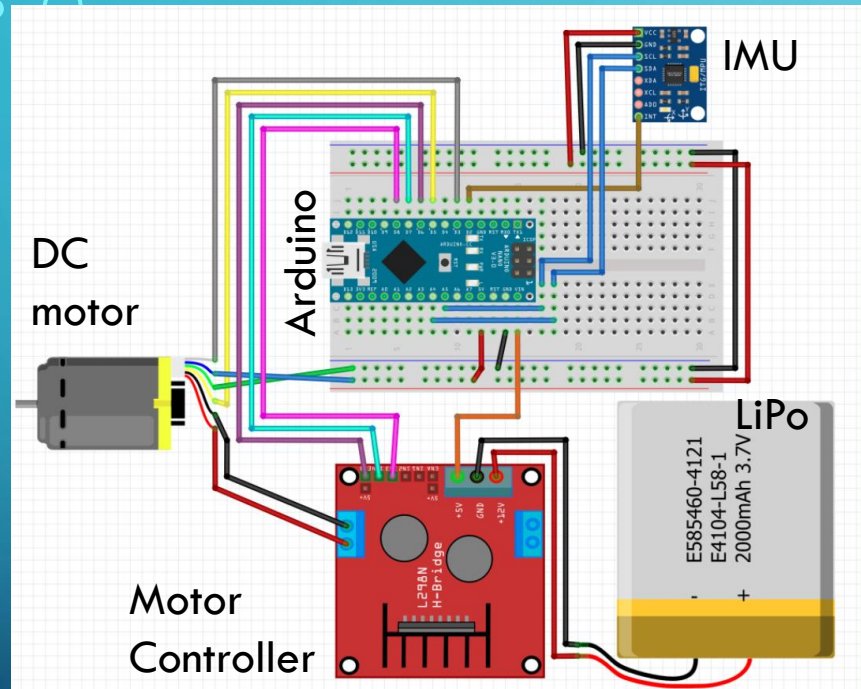
    function WriteDriverVoltage(output, max motor voltage)
    {
        convert output to PWM signal
        set direction of motor
    }
}
```

Pseudocode

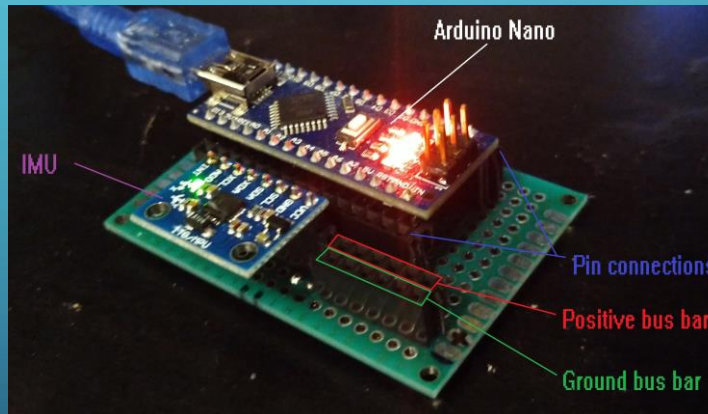


Demo of IMU output

# ELECTRONICS HARDWARE DESIGN



Circuit Schematic



The Arduino and IMU were soldered onto a circuit board to save on space and weight.

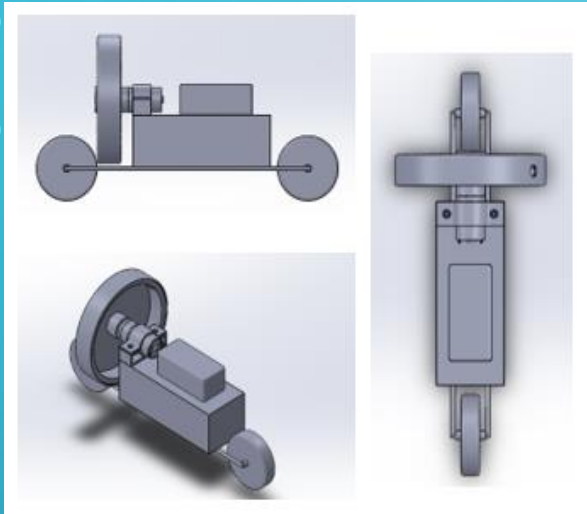
## Primary Components

- **Microcontroller** (Arduino Nano)
- **IMU** (MPU6050)
- **Motor** (12V Brushed DC Motor)
- **Motor Controller** (L298N)
- **Battery** (14.8V 4 Cell LiPo)

## Secondary Components

- **Servo** (5V 180°)
- **Motor Encoder** (included)

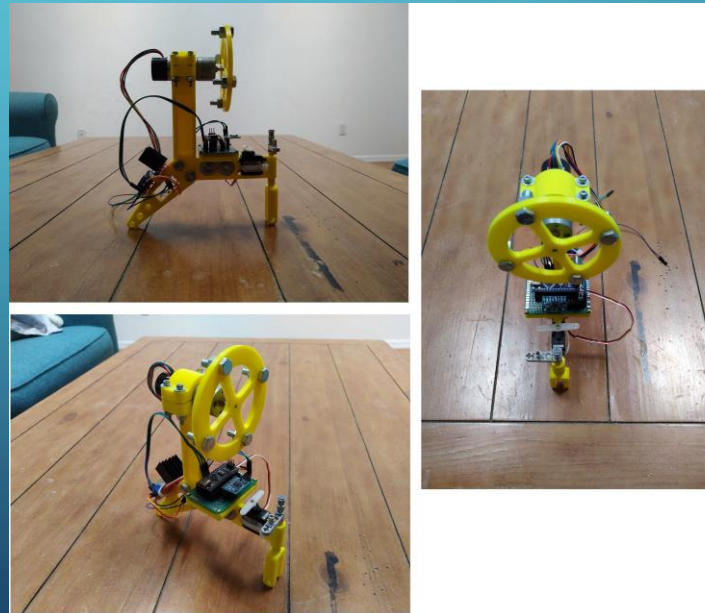
# MECHANICAL HARDWARE DESIGN



1<sup>st</sup> Iteration



2<sup>nd</sup> Iteration



We modeled our prototypes in SolidWorks for 3D printing and importing to Gazebo. A lot of the mounting materials like nuts and bolts were scavenged from around the house.



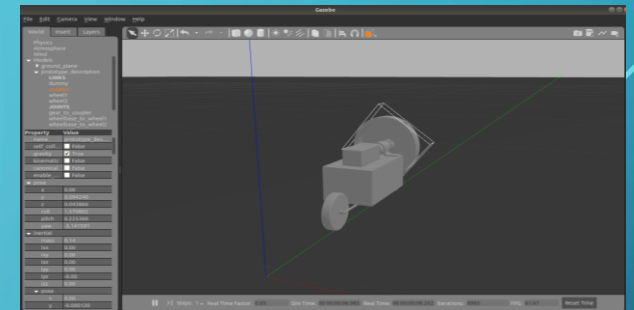
# SIMULATIONS

- Simulation seemed to be the best option when trying to integrate obstacle avoidance and path planning
- Used ROS and Gazebo to implement our code and test new things on an imported 3d model of our prototype.
- The RQT GUI and a little bit of RVIZ proved useful as well

```
1 # gazebo model is the namespace
2 gazebo_model:
3
4 # Publish all joint states .....
5 joint_state_controller:
6   type: joint_state_controller/JointStateController
7   publish_rate: 50
8
9 # Position Controllers .....
10 # This controller receives position and via PID it takes care of the effort to
11 motor_effort_controller:
12   type: effort_controllers/JointPositionController
13   joint: gear_coupler
14   pid: {p: 100.0, i: 100.0, d: 100.0}
15
```

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <!-- This URDF was automatically created by SolidWorks to URDF Exporter!
3 Originally created by Stephen Brawner (brawner@gmail.com)
4 Commit Version: 1.5.1-8-g91665db Build Version: 1.5.7152.31018
5 For more information, please see http://wiki.ros.org/sw_urdf_exporter -->
6
7 <robot name="prototype description">
8   <!-- This dummy link was added bc the base link cannot have inertial
9   properties -->
10   <link name="dummy">
11     </link>
12   <!-- parent link="dummy"/>
13   <child link="body"/>
14   </joint>
15   <link
16     name="body">
17     <inertial>
18       <origin
19         xyz="5.7832E-18 -1.787E-20 1.0452E-36"
20         rpy="0 0 0" />
21       <mass
22         value="0.18" />
23       <inertia
24         ixx="0.0016536"
25         ixy="0"
26         iyy="0.00076023"
27         izx="0"
28         izy="0.0018803" />
29     </inertial>
30   </link>
31
```

```
580 <!-- Add a transmission element to control the motor -->
581 <!-- We will use an effort controller to command a desired force/torque to
582 the joint -->
583 <transmission name="motor trans">
584   <type>transmission_interface/SimpleTransmission</type>
585   <joint name="gear to coupler">
586     <hardwareInterface>hardware_interface/EffortJointInterface</
587     hardwareInterface>
588   </joint>
589   <actuator name="motor">
590     <hardwareInterface>hardware_interface/EffortJointInterface</
591     hardwareInterface>
592     <mechanicalReduction>1</mechanicalReduction>
593   </actuator>
594 </transmission>
595
596 <!-- Add the gazebo ros control plugin -->
597 <!-- The robotNamespace matches in the prototype_control/config/
598 prototype_control.yaml -->
599 <gazebo>
600   <plugin name="gazebo_ros_control" filename="libgazebo_ros_control.so">
601     <robotNamespace>gazebo_model</robotNamespace>
602     <legacyModelNs>true</legacyModelNs>
603   </plugin>
604 </gazebo>
605
606 </robot>
607
```



```
1 main.py
2
3 #!/usr/bin/env python
4
5 import rospy, sys, math
6 from std_msgs.msg import Float64, Float32
7 from sensor_msgs.msg import Imu
8 from geometry_msgs.msg import Twist
9 import pidcontrol as pid
10
11 motorRPM = 600
12
13 class SelfBalance:
14
15   def __init__(self):
16     # Initialize a subscriber node to the IMU data
17     self.sub1 = rospy.Subscriber('/imu', Imu, self.callback)
18
19     # Initialize subscriber nodes for the PID parameters
20     self.sub2 = rospy.Subscriber('/Kp', Float32, self.callback_Kp)
21     self.sub3 = rospy.Subscriber('/Ki', Float32, self.callback_Ki)
22     self.sub4 = rospy.Subscriber('/Kd', Float32, self.callback_Kd)
23
24     # Initialize a publisher node for the motor output
25     # ** Need to publish to a different topic. Not using ros control plugin anymore..
26     self.pub1 = rospy.Publisher('/gazebo_model/motor_velocity_controller/command', Float64)
27
28     # Initialize PID values and a control loop publisher node
29     self.Kp = 25
30     self.Ki = 0.8
31     self.Kd = 0.1
32     self.pub2 = pid.PID_Controller(self.Kp, self.Ki, self.Kd)
33
34   def callback(self, data):
35     # Desired tilt angle
36     setPoint = 0
37
38     # Get the actual tilt angle from the IMU sensor
39     yawAngle = data.orientation.y
40
41     # Calculate PID output to correct undesired tilt
42     output = Twist()
43     output.angular.y = -self.pub2.getCorrection(setPoint, y)
44     # ** assuming output.angular.x & output.angular.z, etc. default to 0..
45
46     # Convert motor RPM to rad/s
47     maxVel = motorRPM / 60 * 2 * math.pi
48
49     # Cap motor velocity
50     if (output > maxVel)
51       output = maxVel
52
53     self.pub1.publish(output)
54
```



# PROBLEMS & FUTURE WORK

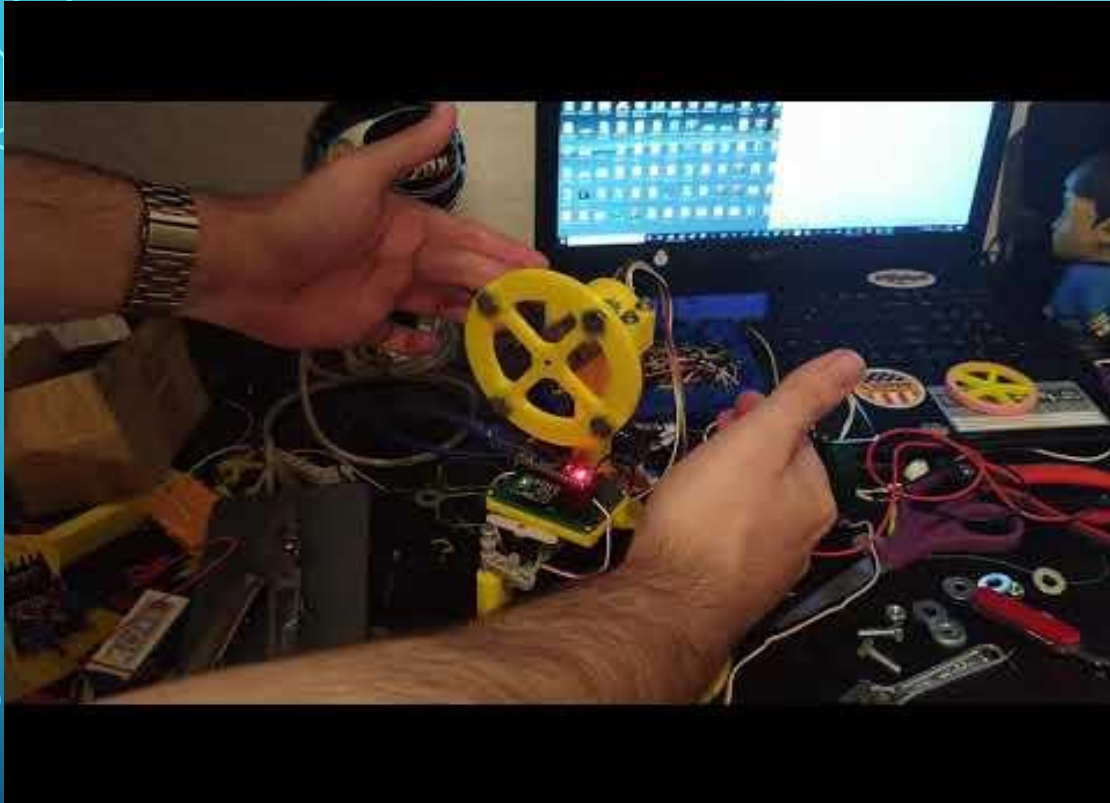
## Problems

- Manual tuning of the PID parameters
- Drift in IMU measurements & inaccurate readings
- Integrating plugins for Gazebo simulation

## Possible improvements

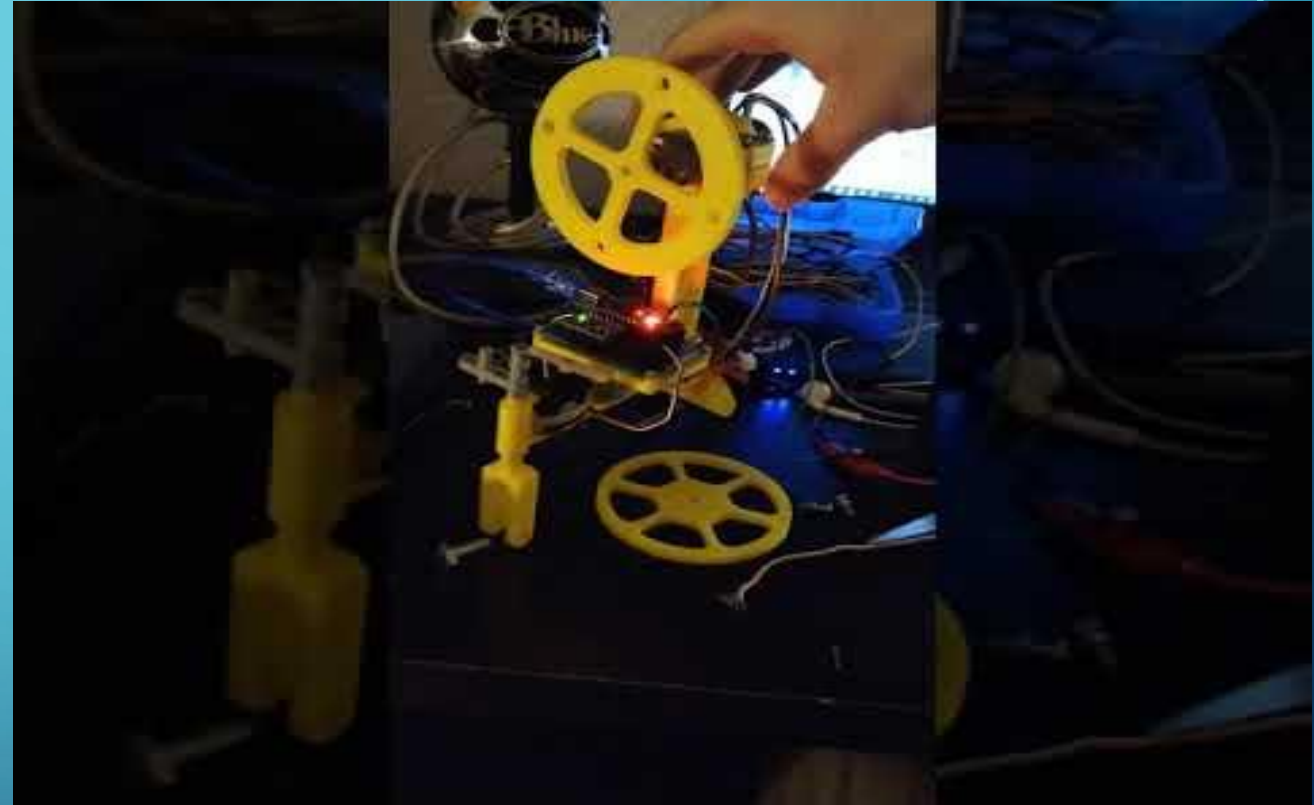
- Use encoder to improve stability
- Pole placement method aka full state feedback
- Fully implement steering capabilities
- Add obstacle avoidance and path-planning
- Add googly eyes

# DEMONSTRATION



**First attempt:**

<https://www.youtube.com/watch?v=dG3BSEhRTi8&feature=youtu.beA&feature=youtu.be>



**Second attempt:**

[https://www.youtube.com/watch?v=HYV\\_trc0QiA&feature=youtu.be](https://www.youtube.com/watch?v=HYV_trc0QiA&feature=youtu.be)

The background is a blue gradient, darker at the bottom. It features abstract line art in a light blue color, resembling circuit traces or data paths, located in the corners. The word "FIN" is centered in a large, white, bold, sans-serif font.

FIN