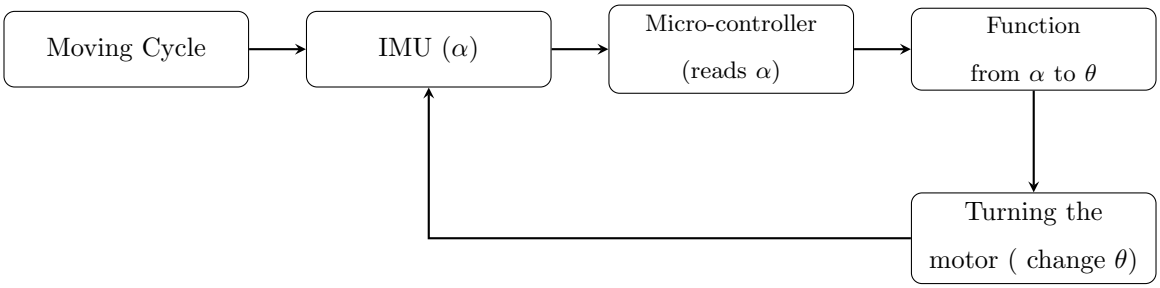
Group Number:		8	
Name:	Anupam Khandelwal	**Roll No.:**	160108008
Name:	Apoorva Kumar	**Roll No.:**	160108010
Name:	Konark Jain	**Roll No.:**	160108022
Supervisor:	Prof. Harshal B. Nemade		
Project Title:	Self Balancing Cycle Bot		

1 Introduction

The aim of this project is to create a self balancing bicycle using weight balancing mechanism. The cycle will initially be a model with a movable frame which can be controlled using a motor to balance the Centre of Mass of the bicycle and keep it balanced by bringing it back to its initial straight position. This bicycle uses an IMU-Inertial Measurement Unit to sense the orientation of main frame of the bicycle and then use that to run a motor using an Arduino to shift the movable body mass and use torque mass balancing to bring the cycle back into starting position.



Where;

α: lean angle of main body of cycle

θ: turning angle of moving mass of bicycle

Figure 1: **Block Diagram**

This system was made to be self sustained and depends on the torque rating of the motor and transfer function used. Our main aim here was to create a platform which can be used to test multiple kinds of motors and different transfer functions.We have initially tried with some types of motors and some transfer functions which will be explained later in the report.

2 Discussion on design

2.1 Mathematical Model

Figure 2 and Figure 3 shows the diagram of the cycle we are trying to balance. As is natural, the base of the wheels points C and D are stationary at any instant of time. The centre of

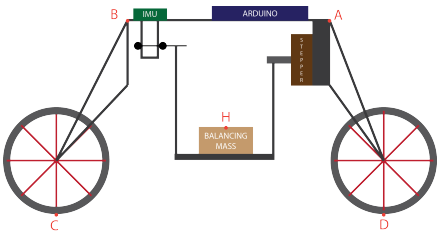


Figure 2: Stick Figure Model

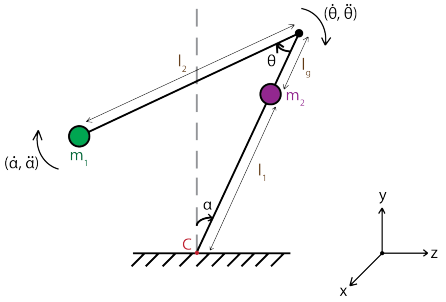


Figure 3: Free body diagram of cycle

mass of the cycle frame (i.e. without the balancing weight) is considered to be located at G and the centre of mass of the balancing weight is considered to be located at H. The problem of balancing a cycle can be considered to be equivalent to the problem of an inverted pendulum being balanced by another simple pendulum attached to the bob of the inverted pendulum. This is depicted as in Figure 2. Consider the lean angle to be denoted by α and the angle the balancing weight needs to move is denoted by θ. α is measured using the IMU sensor. We assume it to be readily available here. Solving for torque balancing using C as the reference point.

$$(m_1 + m_2)gSin\alpha + m_2 l_1 l_2(\ddot{\theta}Cos\theta + \dot{\theta}^2 Sin\theta) = m_2 l_2 g Cos\alpha.Tan(\theta + \alpha) - m_1 l_1^2 \dot{\alpha}^2 \quad (1)$$

$$Approximations: \; Sin\alpha \approx \alpha \;\; Sin\theta \approx \theta \quad ; \quad Cos\alpha \approx 1 - \frac{\alpha^2}{2} \;\; Cos\theta \approx 1 - \frac{\theta^2}{2} \quad (2)$$

$$\theta = \frac{m_2 l_2 g(1 - 0.5 * \alpha^2)\alpha - (m_1 + m_2)g * \alpha - m_1 l_1^2 f^2 * (\alpha - \alpha_{prev})^2}{m_2 l_1 l_2 \omega_\theta^2 - m_2 l_2 g(1 - 0.5 * \alpha^2)} \quad (3)$$

2.2 Physical Model

We built the complete model as shown in Figure 4 which is the almost similar to the our stick model (Figure 2) used for mathematical model. It was built using a set of Mechanix consisting of aluminium bars of strict length and dimensions. The movable weight being used for balancing was made using the same set with added 9v battery to make it have a weight $1/7^{th}$ of main bicycle frame.

This model was chosen because it allows us to make the cycle without adding some extra extended mass to the body of the cycle frame while also allowing the user to ride the cycle. It

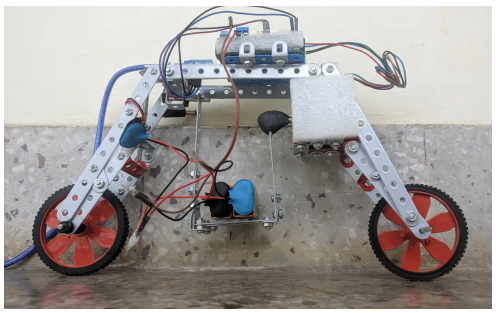


Figure 4: Real World Model

doesn't presently doesn't contain the pedalling system but it can be easily included.
Our model also included an option for steering which can be later used for steering control for better precision and control.

2.3 Circuit Design

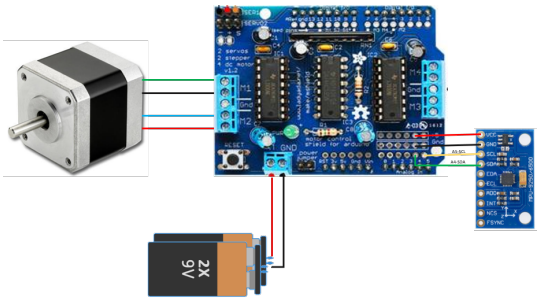


Figure 5: Circuit Diagram

Our fully functional model requires the following equipments and there diagram is illustrated in the Figure 5 above:

- Arduino UNO
- Adafruit Motor Shield v1.2
- NEMA17 4.2kgcm Stepper Motor
- MPU9250
- 2x 9V Battery

The shield is mounted on the Arduino and is a driver with 4 H-Bridges: L293D chip-set which provides 0.6A per bridge (1.2A peak) with thermal shutdown protection, internal kick-back protection diodes. It can run motors with voltage requirement up to 25VDC. We are currently using it with 18VDC powered using two 9v Battery. It has a Arduino package for directly running a stepper motor which we are using to run the stepper motor.
We tested our model with a NEMA17 4.2kgcm Stepper Motor which was readily available and

very commonly used. It has a step angle of 1.8°. Its response time is maximum of 1millisecond for direction change. We can very easily replace this with a better motor with lesser resolution and faster response rate. The response rate of motor was the deciding factor in the frequency of our system.
For IMU we are using MPU9250 which is a 9axis accelerometer-gyroscope-magnetometer.The MPU-9250 is a System in Package (SiP) that combines two chips: the MPU-6500, which contains a 3-axis gyroscope, a 3-axis accelerometer, and an onboard Digital Motion Processor; and the AK8963, 3-axis digital compass. The breakout board we are using the most common with Arduino package to get data input using i2c connections. We are using this MPU as mentioned earlier to measure the lean angle of main frame of bicycle. We chose this model because its more precise with better orientation measurement than MPU6050.
We are usign two 9VDC Battery to run the motor at maximum speed possible without torque loss and slipping.

3 Experimentation

We tested our model with the Stepper Motor NEMA17 which provides a maximum torque of $4.2Kg-cm^2$. We choose this motor because of the high torque and low resolution. Our specific design has the following values for the parameters of the cycle:

Table 1: Parameter Values

Sl.No.	**Parameter Name**	**Parameter Symbol**	**Parameter Value**
1	Mass of Cycle Frame	m_1	0.720 Kg
2	Mass of Balancing Weight	m_2	0.130 Kg
3	Height of Cycle Frame	l_1	0.11 m
4	Height of Rotating Weight	l_2	0.09 m
5	Angular Velocity of Motor	ω_θ	100 RPM
6	Frequency of IMU	f	2 KHz

We then used Equation 3 to calculate the angle that the stepper motor must move in order to provide a balancing action. The stepper motor was rotated at a speed of 100 RPM so that the response time is fast enough but not so fast that it reduces the maximum torque that can be supplied by the stepper. We ran multiple experiments to test the physical stability and we observed that the bicycle could maintain its balance for a short amount of time but because of the non-ideality of the proposed dynamics and physical constraints of electronic equipment - especially the low resolution of the stepper more - it finally destablized after some time.

4 Summary and Future Aspect

We finally conclude that the physical and electronic structure built by us satisfies the need of a self-balancing cycle using mass balancing. This miniature model can be efficiently scaled to standard human bicycle size without any physical constraints. We fell short in this demonstration because of physical constraints of the available electronic equipment. Also to note here is that the the function we have derived for determining θ from α is quiet close to the real value and can be very easily improved upon to get better results. We even found a better motor which can be used and perfectly fits into all the constraints of the system. The link to the motor is **Planetary Geared Stepper Motor**.

Further this model also has provision for steering. No steering control has been implemented but can be done using a servo on the front axle. Steering can be initially used to balance the cycle on a straight path and later use to take turns and still keep itself balanced. Also the modular assumptions of this cycle and the free-body assumptions can be used to solve other physical balancing problems like self balancing robots.

5 List of Component

Table 2: List of Components

Sl.No.	**Item Name**	**Qty**	**Provided** by	**Price(Rs.)**
1	MPU9250	1	Self	200
2	Mechanix	1	Dept	1500
3	Arduino	1	Self	1700
4	Motor Shield V1.2	1	Self	400
5	Stepper Motor	1	Dept	600
6	9v Battery	2	Self	500

6 Arduino Code

Algorithm 1: Arduino Code

```
#include "MPU9250.h"
#include <math.h>
#include <AFMotor.h>

AF_Stepper motor(48, 2);

double theta;
double alpha_prev = 0;
double theta_prev = 0;
double alpha_prev_2 = 0;
double theta_prev_2 = 0;
double acc_y;
double acc_z;

MPU9250 IMU(Wire,0x68);
int status;

void setup() {
      while(1){
           status = IMU.begin();
           if (status>0){
                break;
             }
           else continue;
         }
      motor.setSpeed(100);
    }

void loop() {

  double alpha;
  int stepper_steps;
```

```
33    String stepper_direction;
34    double starttime = millis();
35
36    //SENSOR INPUT AND CONTROL EQUATION
37
38    while(millis()-starttime<50){
39        IMU.readSensor();
40
41        acc_y = IMU.getAccelY_mss();
42        acc_z = IMU.getAccelZ_mss();
43
44        alpha = atan(acc_y/acc_z);
45
46        theta = (0.1147*(1-0.5*pow(alpha,2))*alpha - 8.33*alpha-
47        34848*pow((alpha-alpha_prev),2))/(0.1411-0.1147*(1-
48        pow(alpha,2)));
49
50        alpha_prev = alpha;
51      }
52    //MOTOR CONTROL
53    if((theta-theta_prev)>=0){
54        stepper_steps = (int) abs(theta-theta_prev)*100/PI;
55        motor.step(stepper_steps, FORWARD, DOUBLE);
56      }
57    else{
58        stepper_steps = (int) abs(theta-theta_prev)*100/PI;
59        motor.step(stepper_steps, BACKWARD, DOUBLE);
60      }
61
62
63    // STORING PREVIOUS VALUES
64
65    theta_prev = theta;
66  }
```